National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

UNIVERSITY OF ALBERTA

Quality enhancements in multimedia applications

BY

Shankar Gopalkrishnan ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science.

DEPARTMENT OF COMPUTING SCIENCE

Edmonton, Alberta
Fall 1995

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

THE AUTHOR HAS GRANTED AN IRREVOCABLE NON-EXCLUSIVE LICENCE ALLOWING THE NATIONAL LIBRARY OF CANADA TO REPRODUCE, LOAN, DISTRIBUTE OR SELL COPIES OF HIS/HER THESIS BY ANY MEANS AND IN ANY FORM OR FORMAT, MAKING THIS THESIS AVAILABLE TO INTERESTED PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE IRREVOCABLE ET NON EXCLUSIVE PERMETTANT A LA BIBLIOTHEQUE NATIONALE DU CANADA DE REPRODUIRE, PRETER, DISTRIBUER OU VENDRE DES COPIES DE SA THESE DE QUELQUE MANIERE ET SOUS QUELQUE FORME QUE CE SOIT POUR METTRE DES EXEMPLAIRES DE CETTE THESE A LA DISPOSITION DES PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP OF THE COPYRIGHT IN HIS/HER THESIS. NEITHER THE THESIS NOR SUBSTANTIAL EXTRACTS FROM IT MAY BE PRINTED OR OTHERWISE REPRODUCED WITHOUT HIS/HER PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE DU DROIT D'AUTEUR QUI PROTEGE SA THESE. NI LA THESE NI DES EXTRAITS SUBSTANTIELS DE CELLE-CI NE DOIVENT ETRE IMPRIMES OU AUTREMENT REPRODUITS SANS SON AUTORISATION.

ISBN 0-612-06475-1

Canada

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Shankar Gopalkrishnan

TITLE OF THESIS: Quality enhancements in multimedia applications

DEGREE: Master of Science

YEAR THIS DEGREE GRANTED: 1995

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

(Signed) . . . .
Shankar Gopalkrishnan
3A, 9010 112 St.
Edmonton, Alberta
T6G 2C5

Date: . 18 July, 1995 . .

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Quality enhancements in multimedia applications** submitted by Shankar Gopalkrishnan in partial fulfillment of the requirements for the degree of Master of Science.

Dr. Pawel Gburzynski

W D Grover

Dr. Wayne Grover

Dr. Anup Basu

Date: 18 July, 1995

To God

Abstract

In this thesis, we focus on aspects in enhancing the quality of multimedia applications comprising audio and full motion video data. Distributed/networked multimedia applications involve usage of a communication network to transfer multimedia information to a remote site. Shared media LANs like Ethernet and FDDI offer only limited capabilities for incorporating networked multimedia applications. Hence the interest in using ATM-based switched LANs. ATM has also been adopted as the switching technique in B-ISDN. ATM makes VBR coding of video signals possible. But cell loss seems to be the major drawback in ATM. We propose a layered coding algorithm for motion video. It results in reduction of the subjective deterioration of the resultant image quality in the event of cell loss in the ATM network. This compensatory technique is based on layering in the spatial and frequency domain. It is particularly suited to teleconferencing applications.

Synchronization of multiple data streams in time has been recognized as a significant requirement of future multimedia applications utilizing broadband communication technology. We have implemented a scheme for synchronizing audio and video information for live and orchestrated applications. Two techniques have been considered for intra-medium synchronization while a master-slave-based playback technique is used for inter-media synchronization. The video codec and synchronization scheme have been incorporated in an application which can be used to playback live as well as stored multimedia data.

Acknowledgements

I would like to express my appreciation to my supervisor Dr. Pawel Gburzynski, for his encouragement, guidance and healthy criticism throughout the session. Dr. Dobosiewicz was instrumental in initiating this project. Xerox Corporation's product Netvideo gave the inspiration to focus on a software implementation of a video codec.

I would especially like to thank Aditya, Andrea, "Doctor", Kaladhar, Kannan, Kevin, Lakshmi, Nitin, Raj, Rorogu, Sharad, Srini, Srinivas, Tarvinder, Theodore, Vikas and all those special people at C-DOT and Roorkee who made my stay here memorable.

Contents

1 Introduction **1**

1.1 Introduction to multimedia applications 1

1.2 Types of multimedia applications 2

1.3 Issues in distributed multimedia applications 4

1.3.1 Communication considerations 4

1.3.2 Operating system considerations 10

1.3.3 User behaviour considerations 11

1.4 Thesis layout 12

2 Video over ATM **13**

2.1 Full motion video service over narrowband ISDN 13

2.2 ATM-based B-ISDN 14

2.2.1 VBR coding of video on ATM 15

2.2.2 Problems in ATM 16

2.2.3 Effect of cell loss on video service 17

2.2.4 Compensatory measures to tackle cell loss 18

2.2.5 Layered video source coding schemes 20

2.2.6 Our layered coding approach 21

3 SPAFLAY (SPAtial and Frequency LAYering) **22**

3.1 Video codec design issues 22

3.2 Rationale behind a new layering scheme 23

3.3 **Application considerations in codec design** 25
3.4 **Stages in the SPAFLAY video codec** 27
3.4.1 **Block analysis** 28
3.4.2 **Transform coding** 31
3.4.3 **Layering in SPAFLAY** 35
3.4.4 **Thresholding** 45
3.4.5 **Command refreshment** 45
3.4.6 **Zig-zag sequencing** 46
3.4.7 **Symbol-level RunLength encoding** 46
3.4.8 **Motion detection in SPAFLAY** 47
3.5 **Modes of operation in SPAFLAY** 50
3.5.1 *Prioritized mode* 50
3.5.2 *Normal mode* 52
3.6 **Use of SPAFLAY in a multicast scenario** 53
3.7 **Search in parameter space for SPAFLAY** 54
3.7.1 **Bit-rate-penalty in SPAFLAY's** *prioritized mode* 58
3.7.2 **Image quality in SPAFLAY** 62

4 Media synchronization **77**
4.1 **Temporal synchronization in multimedia applications** 77
4.2 **Accounting for Delay Jitter** 80
4.3 **Synchronization options** 81
4.4 **Intra-medium synchronization** 82
4.4.1 **Blind-Timing and Absolute-Timing** 85
4.5 **Inter-media synchronization** 89
4.5.1 **Framing of data streams** 91
4.5.2 **Temporal presentation control** 98
4.6 **Real-time Transport Protocol (RTP)** 106
4.6.1 **Use of RTP** 107
4.6.2 **RTP Definitions** 107

4.6.3 RTP Fixed Header Fields 109

5 Application functionality 111

5.1 Multimedia on-demand service 112

5.1.1 On-demand service system architecture 113

5.2 Multimedia live multicast service 122

5.2.1 Sending IP multicast datagrams 123

5.2.2 Receiving IP multicast datagrams 125

5.2.3 Client options 126

5.2.4 Client-server architecture 127

5.3 Multimedia local playback service 128

5.3.1 Client options 128

5.3.2 Playback modules 129

5.4 Recording a multimedia session 130

5.5 Rewind and fastforward feature implementation 132

5.5.1 The Rewind feature 133

5.5.2 The Fastforward feature 135

6 Simulating video sessions over ATM 139

6.1 Performance of SPAFLAY over an ATM network 139

6.2 Simulation parameters 141

6.3 Simulation results 142

7 Conclusion and future directions 156

7.1 Future directions 157

List of Figures

1 Classification of multimedia information 3

2 Video transmission over ATM layers 15

3 Threshold for motion detection 30
4 Haar basis vectors for N=8 32
5 Orthonormal Haar matrix for N=8 33
6 Two-dimensional Haar basis matrices for N=8 33
7 Quality-Index (QI) values for Haar Transform coefficients 35
8 ATM Cell 36
9 Spatial prioritization of blocks 37
10 Quality=0: HP and LP Haar Transform coefficients 41
11 Quality=1: HP and LP Haar Transform coefficients 42
12 Quality=2: HP and LP Haar Transform coefficients 42
13 Quality=3: HP and LP Haar Transform coefficients 43
14 Quality=4: HP and LP Haar Transform coefficients 43
15 Quality=5: HP and LP Haar Transform coefficients 44
16 Quality=6: HP and LP Haar Transform coefficients 44
17 Zig-zag sequencing of Haar Transform coefficients 47
18 Reconstructor in SPAFLAY (*normal mode*) 48
19 Reconstructor in SPAFLAY (*prioritized mode*) 49
20 *Prioritized mode* of SPAFLAY 51
21 *Normal mode* of SPAFLAY 52

22 SPAFLAY in a multicast environment 54
23 Image quality using *normal mode* 55
24 Image quality using frequency layering (Quality=0) 56
25 Image quality using frequency layering (Quality=1) 56
26 Image quality (Spatial and frequency layering) 58
27 Bit-rate-penalty in frequency layering 59
28 Bit-rate-penalty: Spatial (20 %) + freq. layering 61
29 Bit-rate-penalty: Spatial (40 %) + freq. layering 62
30 Bit-rate-penalty: Spatial (60 %) + freq. layering 63
31 Case: *Normal mode* 65
32 *Normal mode*: Decompressed image 65
33 Case: Quality = 0 and Spatial(0%) 66
34 Quality = 0 and Spatial(0%): Decompressed image from HP cells 66
35 Case: Quality = 0 and Spatial(40%) 67
36 Quality = 0 and Spatial(40%): Decompressed image from HP cells 67
37 Case: Quality = 1 and Spatial(0%) 68
38 Quality = 1 and Spatial(0%): Decompressed image from HP cells 68
39 Case: Quality = 1 and Spatial(40%) 69
40 Quality = 1 and Spatial(40%): Decompressed image from HP cells 69
41 Case: Quality = 2 and Spatial(0%) 70
42 Quality = 2 and Spatial(0%): Decompressed image from HP cells 70
43 Case: Quality = 2 and Spatial(40%) 71
44 Quality = 2 and Spatial(40%): Decompressed image from HP cells 71
45 Case: Quality = 3 and Spatial(0%) 72
46 Quality = 3 and Spatial(0%): Decompressed image from HP cells 72
47 Case: Quality = 3 and Spatial(40%) 73
48 Quality = 3 and Spatial(40%): Decompressed image from HP cells 73
49 Case: Quality = 4 and Spatial(0%) 74
50 Quality = 4 and Spatial(0%): Decompressed image from HP cells 74

51 Case: Quality = 4 and Spatial(40%) 75

52 Quality = 4 and Spatial(40%): Decompressed image from HP cells . . 75

53 Case: Quality = 5 and Spatial(0%) 76

54 Quality = 5 and Spatial(0%): Decompressed image from HP cells . . 76

55 Reduction of arriving cell delay variance 81

56 Buffering to handle jitter 81

57 Classification of temporal synchronization techniques 82

58 Playout synchronization variables 86

59 Blind-Timing intra-medium synchronization 88

60 Absolute-Timing intra-medium synchronization 88

61 Functionality of Sender-audio-process 96

62 Functionality of Sender-video-process 97

63 Binding audio data units with video frames 97

64 Playback: Stage 1 99

65 Playback: Stage 2 101

66 Functionality of receiver audio process 102

67 Functionality of receiver video process 103

68 Functionality of playbackhandler process (Blind-Timing) 104

69 Functionality of playbackhandler process (Absolute-Timing) 105

70 Basic modules in the client and the server 114

71 Client panel 115

72 Orchestrated playback for multiple clients 119

73 Multimedia multicast service for multiple clients 124

74 Recording a multimedia session 132

75 Rewinding in a multimedia session 134

76 Rewinding in a multimedia session (case 2) 135

77 Fastforwarding in a multimedia session 137

78 Fastforwarding in a multimedia session (case 2) 138

79 Topology of the network 142
80 Cell loss in the ATM network 144
81 Average end-to-end frame delay (Latency) 145
82 Playback cell loss for peak rate access traffic 148
83 Playback cell loss for (Avg. + peak)/2 access traffic 149
84 Switch buffer size=25: Playback cell loss 150
85 Switch buffer size=100: Playback cell loss 151
86 Switch buffer size=200: Playback cell loss 152
87 Switch buffer size=300: Playback cell loss 153
88 Switch buffer size=400: Playback cell loss 154
89 Switch buffer size=600: Playback cell loss 155

List of Tables

3.1 Breakup of high and low priority HT coefficients 41

3.2 Compressed frame size: *Normal mode* 59

3.3 Compressed frame sizes: Frequency layering 60

3.4 Compressed frame sizes: Spatial (20 %) + freq. layering 60

3.5 Compressed frame sizes: Spatial (40 %) + freq. layering 61

3.6 Compressed frame sizes: Spatial (60 %) + freq. layering 61

3.7 Compressing a sequence of images 64

4.8 Skew objectives in multimedia applications 79

Chapter 1

Introduction

1.1 Introduction to multimedia applications

The term multimedia refers to information composed of different data types including text, image, graphics, audio and video. On-going research and development activities have resulted in the ability to process these multiple media by computers. Hardware and software technologies have been developed for processing, digitizing and storing the multimedia information from various devices like video cameras and microphones. In the case of storing the multimedia information, the requirements vary depending on the type of media. Advances have been made in storage technologies and architectures which allow storage of such media information in a workstation. In parallel, there has been considerable progress in the area of communication technology, thereby making it possible to build networks that can support distributed multimedia applications. A distributed multimedia application is characterized by computer controlled generation, processing, communication, storage and presentation of different media information [38].

Multimedia information is generated from devices like video cameras or multimedia servers where information is stored in databases. Multimedia presentation is carried out by bringing information from these sources and delivering them to the multimedia workstation of the user. Presentation of multimedia information involves managing buffer and window space in the user workstation, temporal synchronization of the information delivered, and facilitating user interaction.

Communication requirements [26] for transferring multimedia information over a computer network are more stringent as compared to text files. The network hardware and protocol software should be able to handle these requirements effectively. Supporting any multimedia application over a computer network raises a number of interesting issues to be addressed in areas like communication networks, operating systems and modeling of multimedia systems. Operating system requirements include device driver support for multimedia devices, file system support for handling large multimedia files and real-time scheduling support. Modeling techniques are needed for characterizing the different multimedia application requirements including dynamic user behaviour. Dynamic participation of the user in a multimedia presentation may modify the communication and operating system requirements.

1.2 Types of multimedia applications

Multimedia applications may be classified depending on the mode of generation of information, on the time domain of information and on the nature of information transfer.

Fig. 1 shows the classification of multimedia information. Multimedia information can be generated either through devices like video cameras and microphones or through accessing stored information in databases/files. A multimedia application is termed *orchestrated* if the capture and/or generation of information is done by retrieving stored information. On-demand HDTV server and multimedia database applications fall under this category. These applications typically access stored information in large optical disks either locally or in a remote system. If the multimedia application processes information from devices like video camera, microphone or keyboard, then it is termed as a *live* application. Teleconferencing and panel discussion type of multimedia applications fall under this category.

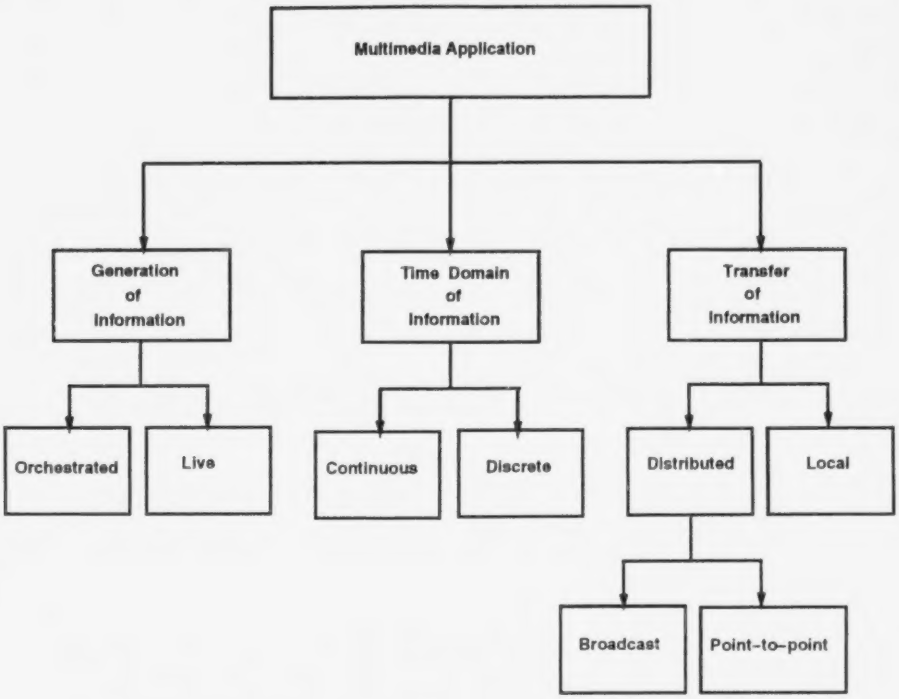

Multimedia Application
Generation of Information
Time Domain of Information
Transfer of Information
Orchestrated
Live
Continuous
Discrete
Distributed
Local
Broadcast
Point-to-point

Figure 1: Classification of multimedia information

In a similar manner, multimedia information can be classified into two categories on the basis of the time domain: Discrete or time independent media and Continuous or time dependent media. Discrete media such as text, graphics and image have no real-time demands. Continuous media like audio and video include real-time requirements. In the case of continuous media, the information becomes available at different time intervals. The time intervals can be periodic or aperiodic depending on the nature of the media. Video and audio information are mostly periodic in nature. Orchestrated and live multimedia applications can be composed of both discrete and continuous media.

In live applications, information is acquired live from sources (video camera, microphone, etc.) and hence the temporal relationships of the events in the media are implied. This temporal relationship is related to the sampling rate used for the media. For video, it could be 30 frames/second (fps) and for audio, the rate at which information is acquired varies from 16 to 64 Kbps. For an orchestrated application, the temporal relationships for different media have to be explicitly formulated and stored along with the media information in databases.

A multimedia application can be *local* or *networked/distributed.* A local multimedia application gets information from sources present on the same workstation/system. A networked or distributed multimedia application involves usage of a communication network to transfer multimedia information to a remote site. Commercial success of multimedia depends on the ability to support networked applications. Most applications in the corporate world are now networked and corporations expect the same functionality in regard to multimedia applications.

1.3 Issues in distributed multimedia applications

In the design and implementation of distributed multimedia applications, the following issues must be considered:

- Communication considerations
- Operating system considerations
- User behaviour considerations

1.3.1 Communication considerations

Communication requirements for a distributed multimedia application depend on the type of the application and the types of media composing the application. Continuous media, like video, demand high bandwidth, low end-to-end delays (latency), low delay variations (jitter) and low packet loss for transferring the required information. Discrete media like images do not have such stringent requirements, but they are more sensitive to packet loss than video. Digital video in uncompressed form needs maximum bandwidth. Different video digitization and compression techniques have been developed to reduce the bandwidth requirements. Apart from requirements like end-to-end throughput guarantees, distributed multimedia applications require a number of simultaneous active network channels for transferring different media: video, audio, image and text. There is also need for synchronization across different media. For example, the video information has to be synchronized with the corresponding audio [26]. Hence, the computer network may have to provide services such as guaranteed end-to-end throughput, delay, Delay Jitter and inter-media synchronization to the application. These parameters referred to as QoS parameters, have to be guaranteed by the network service provider.

Quality of Service (QoS) considerations:

The Quality of Service offered by the network to a multimedia application can be characterized by parameters like traffic throughput, transmission delay, Delay Jitter, transmission reliability and inter-channel relationship.

- **Traffic throughput:** It is the amount of data that will be sent through the network specifying the traffic communication needs in terms of the bandwidth required.
- **Transmission delay:** This is the delay that the transmitted data will suffer through the network. It is expressed in terms of an absolute or probabilistic bound.
- **Delay Jitter:** The variable delays incurred by packets over a network give rise to Delay Jitter. A bound on jitter or delay variation is often specified.
- **Transmission reliability:** It is related to the buffering mechanisms involved in data transmission along the network. Because of the limited size of these buffers, during traffic congestion, there is buffer overflow resulting in packet loss. A probabilistic bound on such kind of losses influence resource allocation during connection set-up.
- **Inter-channel relationship:** In a multimedia application, a number of network channels are required simultaneously for transferring different media ob-

jects. In some cases, inter-channel synchronization has to be provided when the channels are to be used for transferring media like audio and video. Relationship among channels can also be specified in terms of inter-channel bounds on QoS parameters (bounds on Delay Jitter, in case of audio and video channels) or multicast relationship. The process of interaction between the application and the network service provider to determine *preferred* and *acceptable* QoS values is termed as *negotiation of the QoS parameters.* The commitment by the network service provider can be hard or soft. In case of hard guarantees, the network commits to offer service, whose quality is precisely specified through a number of traffic and performance parameters. In case of soft guarantees, no strong commitment is made by the network service provider. The application should be able to handle the dynamic modification of the offered quality of service.

Network protocol considerations:

The QoS requirements discussed above, have to be supported by the network protocol providing the communication services. Network protocols must be designed to support the following:

- **Transmission reliability:** Retransmission of lost data may not be required for live multimedia applications where audio and video are transmitted in real-time. In this case, error handling aspects are left to the application. But for a media like text, we need reliable transmission. Thus a combination of reliable transmission and real-time transmission has to be provided by the network protocol.

- **Multicast service:** Multicast service is needed by most *live* multimedia applications. The network service provider should offer a flexible addressing mechanism to allow identification of groups of related network channels. Dynamic joining and leaving of multicast groups should also be allowed by the network service provider.

- **Flow control:** The network protocol has to employ different types of flow control mechanisms for different applications. For example, for transaction based applications like file transfer, sliding-window flow control mechanisms are suitable. But for continuous media like video, which have a predetermined rate of transmission, a rate-based flow control mechanism is preferred. In this scheme, the transmission rate is negotiated at connection set-up time and the sender can transmit data without waiting for acknowledgements from the receiver. The application must be able to choose the mechanism during network channel establishment time.

- **Synchronization:** Synchronization is required by multimedia applications [37] for coordinating the presentation of related media information like audio and video. The required synchronization can be either supported by the network service provider (e.g., Transport Layer) or be realized by the application themselves. If the application is to realize synchronization, then it has to be done by enforcing strict bounds on delay and Delay Jitter requirements at the Transport Layer for the different streams. This option can reduce the complexity of the network service provider implementation. In this thesis, we have considered schemes for synchronization at the application level. The other alternative is to provide synchronization as part of the network service. In this case, the network service provider must have some knowledge about the data.

- **Negotiation of QoS:** The offered Quality of Service must be negotiable. The negotiation can be between the application and the network service provider or between the source and the destination applications. In this thesis, we use negotiation of QoS between the source and the destination applications. The network service provider should provide for negotiation so that applications are fully aware of the QoS parameters used. The offered QoS might degrade during the connection time due to varying load conditions in the network. The network service provider has to provide services for handling degradation of Quality of

Service.

Network bandwidth considerations:

The network service provider provides a Quality of Service which depends on the network hardware platform that is available to it. The following characteristics of the network medium have a distinct bearing on the type of support provided by a network protocol.

- Network bandwidth
- Network access control mechanism
- Priority control schemes for network access

Throughput guarantees by the network service provider depends on the bandwidth offered by the network hardware. The network bandwidth ranges from medium speed (operating at a few Mbits/s) to very high speeds (operating at several hundreds of Mbits/s). The delay guarantees depend on the medium access control methods and the availability of priority control schemes to access the physical medium. In this section, we look at a few commonly used networks [24] and the type of guarantees they can offer to networked multimedia applications.

Medium speed LANs:

Ethernet, Token Ring and Token Bus fall under medium speed LANs. Ethernet (IEEE 802.3) has a bus topology which offers a maximum bandwidth of 10 Mbits/s over a coaxial cable using CSMA/CD protocol to gain access to the bus. The disadvantage of Ethernet is in the CSMA/CD access mechanism, which does not guarantee an upper bound on network access delay. Also priorities cannot be assigned among stations waiting to transmit information. Hence, Ethernet based networks cannot guarantee absolute bounds on end-to-end throughputs and delays.

Token Ring (IEEE 802.5) networks permit tokens to circulate around the network. A station can transmit for a fixed duration when it can catch hold of a free token. Token Ring networks permit priorities to be assigned to stations to capture a free token. Hence, we can give more precedence to stations which transmit continuous media information. As a result, in token-based networks, guarantees can be provided for end-to-end throughput, delay and Delay Jitter.

Fibre Distributed Data Interface (FDDI):

An FDDI network provides a throughput of the order of 100 Mbits/s over an optical fibre medium. The access protocol for FDDI is based on the Token Ring, providing synchronous as well as asynchronous access to the network. As in Token Ring, we can provide an upper bound on access delay. FDDI network offers a guaranteed throughput for synchronous transmission, which can be used for voice and video information.

ATM-based networks:

It is felt that networks based on shared media will not be able to supply large aggregate bandwidths needed by future multimedia applications. Switch-based digital networks seem the most appropriate technology to meet the new multimedia challenges. Hence the growing interest in Asynchronous Transfer Mode-based LANs. Asynchronous Transfer Mode (ATM) relies on Asynchronous Time Division Multiplexing. It is the transfer mode for B-ISDN. **Chapter 2** presents an overview of ATM and the reasons why it is particularly attractive to multimedia applications.

From the discussion presented above, we see that the QoS offered by the network to a multimedia application depends on the network protocol software and hardware employed. Choice of a suitable network protocol has a tremendous bearing on the performance of networked multimedia applications.

1.3.2 Operating system considerations

The operating system has to satisfy varied requirements to support a distributed multimedia application. The demands exercised by multimedia applications are in:

- **Handling large multimedia files:**

  The multimedia file system must be able to handle huge files (of the order of gigabytes). Most of the existing storage architectures allow unconstrained allocation of blocks on disks. Since there is no constraint on the separation between disk blocks storing a chunk of digital video or audio, there is an appreciable variation in the access and latency times. Contiguous allocation of blocks can guarantee continuous access, but has the familiar disadvantage of fragmentation of useful disk space. Constrained block allocation can help in guaranteeing stricter bounds on access times without encountering the above problems. For constrained allocation, factors like size of the blocks (granularity) and separation between successive blocks (scattering parameter) have to be determined for ensuring guaranteed bounds.

- **Real-time scheduling:**

  Real-time requirements of multimedia applications arise mainly in the form of:

  - Handling Network traffic
  - Handling multimedia devices

  Transferring digital video and audio information over a computer network implies large bandwidth and low delay requirements. To satisfy these requirements, the operating system should be able to handle the network traffic. Multimedia devices like video camera and microphone generate information at regular intervals and hence these devices have to be serviced periodically with real-time restrictions. Overhead in the I/O mechanisms can result in timing errors and loss of data even though the hardware is able to handle the high data rates.

- **Device driver support:**

  To support multimedia applications, an operating system should provide support to handle different types of devices : video camera, microphones, speakers, etc. The operating system should provide a uniform interface to the devices so that multimedia applications can use them with minimum modifications.

  In **Chapter 4**, we discuss the need for real-time scheduling for supporting reasonable playback of a multimedia application.

1.3.3 User behaviour considerations

User in a multimedia application can participate dynamically by giving different inputs The nature of the user participation depends on the type of multimedia application: orchestrated or live. User going through an orchestrated presentation can participate by giving inputs like pause, review, forward, freeze and restart of presentation. User inputs to an orchestrated multimedia presentation modify the network Quality of Service (QoS) requirements, by modifying the temporal relationships between the objects and the size of the objects. Inputs like skip, reverse presentation and navigate modify the presentation sequence and hence the instantaneous QoS requirements might be modified.

In an interactive multimedia presentation, user can participate by giving inputs like freezing and restarting the presentation. The user can dynamically join or leave the presentation. He can also dynamically introduce new devices forcing more communication channels to be established and thereby modifying the QoS requirements. The change in the number of active channels will also change the operating system requirements.

In **Chapter 5**, we will look into the ways in which the user can participate in our application.

1.4 Thesis layout

This thesis focuses on aspects in enhancing the quality of a multimedia application involving full motion video and audio. In keeping with the various types of multimedia applications discussed in this chapter, we have implemented an application which can be used in various modes. It can perform as a *live* application, an *orchestrated* application, both locally and over a network. It can operate in an *interactive* and *static* mode. Distribution of multimedia information has been considered using both *unicasting* and *multicasting*. It supports local as well as remote recording of multimedia information. The detailed functionality of the application is presented in **Chapter 5**.

The application was developed to serve as a testbed for our video codec named SPAFLAY. This codec has been designed and implemented with focus on ATM networks. The rationale for our interest in ATM networks is explained in **Chapter 2**. The codec uses a layering scheme to guarantee quality of a video image in the face of loss in the ATM network. Though the full potential of the codec is tapped by teleconferencing applications, the range of use of the codec is by no means restrictive. The layering scheme also shows immense potential in the scenario of multicasting video. The codec (SPAFLAY) is discussed in detail in **Chapter 3**.

Synchronization of audio and video schemes form an important part of the application. We have considered techniques for media synchronization for both *live* and *orchestrated* applications. **Chapter 4** gives an account of the synchronization schemes used in the application.

In **Chapter 6**, we present the simulation results of video sessions over an ATM network.

Chapter 2

Video over ATM

2.1 Full motion video service over narrowband ISDN

Narrowband Integrated Services Digital Network (ISDN) cannot support full motion video service even by means of advanced data compression and signal processing techniques. The reasons for this is due to its limited and fixed bandwidth and the switching protocol that it uses. Narrowband ISDN employs circuit switching in the synchronous transfer mode (STM) for continuous media traffic like audio and video. Due to the inhomogeneous nature of video sources and the inherent variations of activity from scene to scene, the information rate after data compression tends to be highly variable and unstable. This inhomogeneity in the compressed video data is not exploited by STM-based fixed rate transmission. In the STM-based fixed rate transmission environment of narrowband ISDN, this inhomogeneity and variation are partially compensated for by buffering and matching to the characteristics of the constant rate channel. This causes certain delay for video sources with low motion and causes picture degradation for video scenes with active motion. The fixed-rate channel also puts some restriction on the capability of data compression schemes, since the compressed data rate for the busiest video scene condition has to be taken into account. ATM-based B-ISDN has emerged to alleviate these limitations in narrowband ISDN.

2.2 ATM-based B-ISDN

In Broadband Integrated Services Digital Network (B-ISDN), all different types of traffic including voice and video are packet switched operating on the cell-relayed ATM format. The adoption of Asynchronous Transfer Mode (ATM) as the switching protocol and the introduction of synchronous optical network (SONET) as the transmission standard greatly facilitate the implementation and commercialization of the B-ISDN. The flexibility and the high speed of the ATM switch enable the B-ISDN to efficiently transport and switch services of different characteristics like data, voice, audio, graphics and image as well as video. The main asset of ATM is its capability to absorb the variations of different types of traffic, and hence allow the integration of sources with different bit-rate and statistical characteristics. The bandwidth of a B-ISDN is greatly expanded with the standardization of the high speed transmission protocol of SONET. This gives the possibility of switching and transmitting full motion video signals which narrowband ISDN could not achieve. An ATM-based B-ISDN layered protocol and the corresponding functions for video transmission [40] are illustrated in Fig. 2.

The ATM-based network consists of a user independent network core (ATM layer) and the user dependent network interface (ATM adaptation layer). The network core provides integrated cell multiplexing and switching services regardless of the incoming source formats. On the other hand, the ATM Adaptation layer performs cell assembly/disassembly and the individual protocol on each service, such as lost cell handling for voice/video service or cell retransmission for data service. In the User Plane Layer (UPL), video signal formatting, video encoding/decoding, high-level signal processing and end-to-end error recovery are performed. The Control Plane Layer (CPL) is responsible for end-to-end signalling and routing. An ATM cell

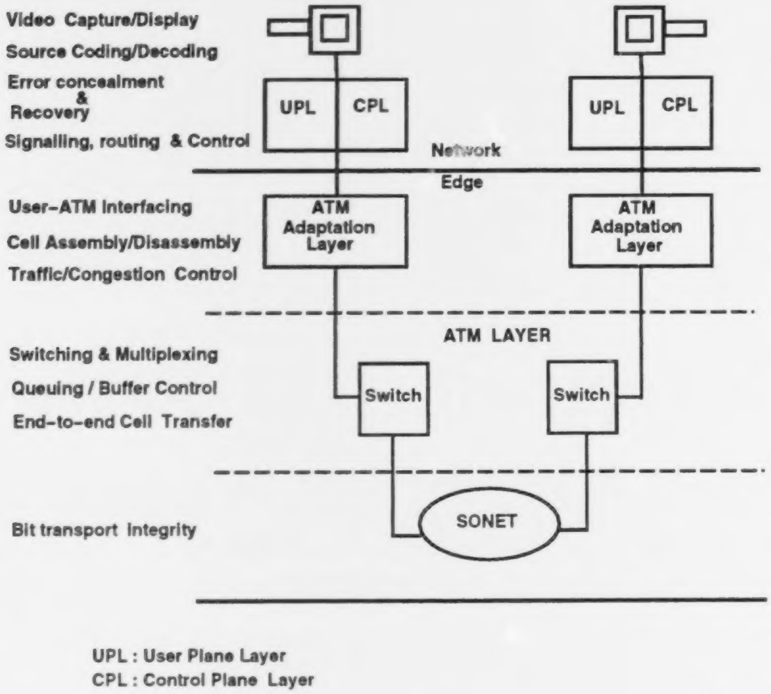

Video Capture/Display
Source Coding/Decoding
Error concealment & Recovery
Signalling, routing & Control
User–ATM Interfacing
Cell Assembly/Disassembly
Traffic/Congestion Control
Switching & Multiplexing
Queuing / Buffer Control
End-to-end Cell Transfer
Bit transport Integrity
UPL
CPL
UPL
CPL
Network
Edge
ATM Adaptation Layer
ATM Adaptation Layer
ATM LAYER
Switch
Switch
SONET
UPL : User Plane Layer
CPL : Control Plane Layer

Figure 2: Video transmission over ATM layers

is formed in the adaptation layer and is queued and multiplexed to form a SONET frame in the physical layer.

2.2.1 VBR coding of video on ATM

ATM makes variable bit-rate (VBR) coding for video signals possible. In VBR coding schemes [16, 27, 39], no buffer is needed as in constant bit-rate schemes and a time-transparent or rate-free transmission can be realized. The fixed rate coding or the fixed-bandwidth system is controlled by the coding rate, while the quality varies with the activity of the instantaneous scenes. In VBR coding, data rate may vary drastically, but the picture quality is kept constant. In addition, the packetization delay is decreased because a fixed-rate buffer is not used in VBR coding. ATM favours VBR coding [7, 20] because it consists of a stream of small cells that are statistically multiplexed regardless of their bit-rate and content. A statistical gain is obtained by taking advantage of the burstiness and variations of the compressed video source traffic. From the user's viewpoint, the advantages of VBR coding due to the introduction of ATM include:

- Format-independent, rate-free and time-transparent transmission. This is due to the fact that we do not need a buffer as in constant bit-rate schemes. This also implies that we don't have to wait for the buffer to get filled before transmission which is true in constant bit-rate schemes.
- Reduction of end-to-end delay as compared to constant bit-rate schemes which rely on the buffer to get filled before transmission.
- Consistent video quality even for the active motion area/frames.
- Quality control by the user instead of rate control by the channel.

From the network point of view, the advantages of ATM include:

- Dynamic bandwidth allocation [28].
- Easy multimedia (data/voice/image/video) integration.
- Network delay/throughput performance improvement.

2.2.2 Problems in ATM

While VBR coding in the ATM environment provides the network with a number of advantages over fixed rate coding in STM circuit-switched networks, it also causes problems that do not exist in STM networks. These problems are due to cell loss and Delay Jitter. Cell loss seems to be the major drawback for VBR video coding in ATM networks [40] due to the high bit-rate and bursty nature of the traffic. In ATM, cell loss is basically of two types: one type occurs randomly due to transmission errors while in the other type, cells are dropped deliberately by the network traffic control protocol in case of network/buffer overflow.

In an ATM network, a cell can be lost due to the following reasons:

- **Channel error:** This cell loss is due to a random error during transmission. Communication channels are subject to different impairments. If an error occurs in the address field of an ATM cell, the cell will not be delivered to the correct destination. This cell is considered to be lost.

- **Limitation of network capacity:** This cell loss is introduced deliberately by the network control protocol [32] in case of network congestion/buffer overflow. If the network is congested due to heavy traffic, the network congestion control protocol will be forced to drop cells.

In ATM, cell discarding can occur on the transmitting side and on the receiving end.

- **Cell loss at the transmitting end:** Cells are discarded by the sender if the number of cells generated are in excess of transmission capacity. If the incoming traffic exceeds the transmission capacity, the sender could be informed by the network traffic control protocol to reduce the traffic flow or switch to a lower-grade service mode.

- **Cell loss at the receiving end:** If the error occurs in the cell header [25] especially in the address field, the cell may get misdelivered or go astray in the network. In the receiver, if the cell is not received within the maximum time-out window, the cell is considered to be lost.

2.2.3 Effect of cell loss on video service

ATM consists of a stream of small fixed-size cells. Each cell consists of a 5 byte header and a 48 byte information field. The loss of a cell leads to the loss of 384 consecutive bits, which may cause severe degradation in picture quality. Network congestion may cause a few consecutive cells to be lost. Inter-frame coding (a video frame coded with respect to other frames) which is used frequently in video encoding is propagative in nature. Hence, cell loss will generally affect a number of subsequent frames.

Also, VBR coding techniques exploit temporal redundancy (that exists between video frames) and spatial redundancy (that exists within an image). It is this propagative nature that makes these VBR systems sensitive to channel errors. Thus cell loss is a major problem encountered in VBR coding in the ATM environment. Cell loss may cause direct picture quality degradation. It may also cause problems like *cross scene* effect. This is due to misdelivery of mis-routed cells that are associated with corrupted headers. If a packet is delivered to another user due to channel error, the end user might get the wrong packet. The cell loss problem in an ATM network is a hot issue in current research. But there seems to be no complete preventive scheme to solve this problem as of now. Hence, we are forced to look into compensatory measures to reduce the effects of cell loss on VBR transmission. In the next section we look at the various schemes that have been proposed.

2.2.4 Compensatory measures to tackle cell loss

The cell loss which occurs due to random error in an ATM network can be controlled by employing forward error correction (FEC) code or by improving the channel error performance. An 8-bit error check sequence is used to reduce the possibility of cell misdelivery and *cross-scene* due to channel errors. This error is of little concern to us.

But the performance degradation due to cell dropping is more serious. It is not possible to prevent this error as we said earlier, due to the contradiction between the available bandwidth and the higher bandwidth requirement in peak traffic. Hence, we have to take some compensatory measures. This motivates consideration of robust VBR image compression algorithms capable of delivering viewable (but degraded) pictures under occasional high cell loss conditions expected in ATM networks. In these compensatory schemes, the purpose is to reduce the performance degradation on the image quality and make the objective impairment subjectively imperceptible. These schemes are also useful to compensate for the random cell losses due to channel error.

Automatic Repeat ReQuest (ARQ) scheme: The general concept of automatic repeat request [5] is to detect frames with errors or the loss of frames and request the transmitting module to repeat the information in those erroneous or lost frames. ARQ schemes are ill suited for video transmission although they are adequate for data services and other non real-time communication applications. Retransmission makes the network more congested, if the cell loss was due to traffic overflow.

Error concealment by command refreshment: Due to the propagative nature of VBR coding schemes, errors spread from one ATM cell to another and may lead to an interruption in the normal transmission process. This problem can be mitigated by sending a frame coded with respect to itself alone periodically. This refreshes frames periodically and error propagation is restricted to certain video segments and frames. This scheme is called the *enforced command refreshment* scheme and is used in many video transmission systems.

Layered source coding: In this scheme, we classify video information into different classes with different priorities. The objective is to restrict cell dropping to low priority cells and guarantee timely and error-free transmission of high priority cells, thereby minimizing the degradation of the picture quality when the network becomes congested. In layered video transmission, video quality degrades gradually with the cell loss rate where only the cells in the least important layer are discarded. Layering of compressed information into high priority (HP) and low priority (LP) tiers [16, 18, 19, 21] makes it possible to guarantee a minimum level of subjective performance at the decoder under conditions of extreme congestion, and is consistent with the 2-priority link service planned for ATM. Layered coding schemes have emerged as the most popular coding scheme for packet video applications.

2.2.5 Layered video source coding schemes

Layered coding schemes can be broadly divided into those which operate on the spatial domain and those which function on the frequency domain. In the spatial domain, a technique called Feature Plane Separation is used. In the frequency domain, we use transform coding schemes.

Feature Plane Separation (FPS): This is a subjective separation scheme depending on the information interest for specific applications. FPS is a content driven representation of video signals. This is normally done in the spatial domain where we separate the image into areas of varying interest. For example, in a radiological environment, the tumor information of a chest x-ray might be more important to the physician than the background. In that case, we can send the cells from the region of interest with a higher priority than cells which comprise the background. Thereby, even the loss of the low priority cells would not significantly alter information in the decoded picture.

Transform coding: Natural image/video scenes contain a lot of redundancy in the spatial/temporal domain. The purpose of the transform coding approach is to map these highly correlated image samples to another domain in which image coefficients become statistically independent. We use transforms like Discrete Cosine Transform (DCT) [29], Haar (HT) [6], etc. A video signal can be considered as a sequence of images. An image compression scheme can be applied to digital video by applying it to individual video frames. As there is a lot of redundancy in the temporal domain (frame-to-frame), inter-frame coding makes substantial reduction in the data rate. Inter-frame coding can be done using conditional replenishment. Conditional replenishment is an inter-frame predictive coding scheme based on coding and transmitting the differences between the present frame and the previous frame.

In Combined Transform Coding (CTC) [14], the transform coefficients are divided into two planes: The Upper Image Plane (UIP) and Low Image Plane

(LIP). The UIP contains the most important information, and error-free and timely delivery of UIP cells should be guaranteed. The LIP contains less important information, and the delivery of LIP cells is not always guaranteed. In the case of congestion, the cells comprising LIP may be lost. The advantage of this method is that even with loss of LIP cells, we will still be able to obtain a picture of reasonable quality after decoding using solely the UIP information.

2.2.6 Our layered coding approach

In this thesis, we implement a layered coding scheme using a combination of Feature Plane Separation and Combined Transform Coding techniques. It is called SPAFLAY (SPAtial and Frequency LAYering). The approach is particularly suited to applications like teleconferencing and distance education.

In these applications, there exists an area (areas) which is (are) of greater interest than the remainder of the picture. In this area (fovea) more detail is required. The outer regions (periphery) are often of secondary importance, and thus less detail is required. Also, motion in the video frames is clustered normally around the fovea. We use a non uniform image subtraction scheme to detect the areas of motion. This scheme is sensitive to areas of motion around the fovea and less to regions in the periphery. We also use the Feature Plane Separation approach to categorize information around the fovea as high priority (HP) and information away from the fovea as low priority (LP). The Combined Transform Coding scheme is again used to split the transform coefficients into two priorities. This layered coding scheme is followed by thresholding the transform coefficients and entropy coding.

This coding strategy is intended to provide robust delivery of video under possibly high ATM cell loss conditions by producing output codec data partitioned appropriately for prioritized transport over ATM. This codec forms the subject matter of the next chapter.

Chapter 3

SPAFLAY (SPAtial and Frequency LAYering)

3.1 Video codec design issues

Multimedia applications are computationally intensive. Users have to invest in costly studios and codecs, and must use dedicated network connections. However, the growing computing power of RISC workstations open possibilities of software implementations of video codecs. Hence, one of the aims of this thesis is to design and implement a software video codec. The performance, using the processors of the day, is already quite good. The performance of CPUs is expected to multiply in the coming years. This will definitely result in an increase in the performance of software codecs [17].

Since our codec can be integrated in a regular software environment, it offers new networking possibilities. We have used the codec over a UDP (User Datagram Protocol) transport protocol to test full motion video performance and quality. The codec is embellished with features which make its performance particularly attractive over ATM networks. It uses a unique layered coding algorithm suited to ATM networks. The codec has been tested over regular internet connections to estimate the video image quality that we may achieve over ATM.

Our basic codec design goal is to produce a robust, VBR, compressed video stream with HP/LP (HighPriority/LowPriority) layering, while incurring a relatively low bit-rate overhead compared to one-layer approaches. Layering is done in the spatial and frequency domain. Hence the name SPAFLAY (SPAtial and Frequency LAYering) for the codec. In general, layering may involve some inefficiency in the coding algorithm itself due to an increase in the redundancy of the layered structure. We need to determine a procedure to partition the video stream into HP/LP portions with appropriate bit-rate properties. It is desirable to produce a HP layer which is a relatively low fraction of the total bit-rate (since high priority transport is expensive) and is characterized by low bit-rate variance, but is sufficient to provide a reasonable image quality level even during heavy cell loss. A related consideration is the quality of the decoded image in the presence of cell loss. It should be acceptable as per the anticipated needs of the application. This will have a bearing on determining the partition of the HP/LP portions of the video stream.

3.2 Rationale behind a new layering scheme

The multimedia application that we have designed and implemented is intended to be used primarily for teaching purposes and teleconferencing. In these applications, we visualize a typical *talking head* scene for most part of the time. This provides an ideal fovea location as discussed in the last chapter. The background for the talking head scene is of little importance to us.

ATM networks have a two-level priority scheme which makes robust video coding possible. In the event of cell loss in the ATM network, we would prefer cell loss to be restricted to the background information of the scene. This would cause little subjective degradation in image quality. Accordingly, it would be reasonable to segment the scene into two parts: the area of interest would be sent as high priority ATM cells, while the background scene information would be tagged as low priority cells. Thus we can use the concept of Spatially Varying Sensing [3] to determine the partition of HP/LP portions of the video stream.

Another approach which could be used for all scenes, irrespective of the presence of an area of interest is splitting of transform coefficients. Transform coding using Haar Wavelet has the advantage of separating compressed information into frequency components which have graduated subjective importance, and hence are amenable to prioritized layering for ATM. For Haar Wavelet-based compression techniques, a natural layering method is to transmit key header information and the first few low frequency coefficients in HP, while sending the remaining high frequency coefficients in LP. Using only the HP (low frequency coefficients) data, we can reconstruct the entire image with reasonable quality. Loss of high frequency components will only result in a decrease in the details and sharpness of the image.

In our model, we use a combination of these two approaches. It is particularly beneficial to fovea-oriented scenes. At the start of the video session, the pixel at the centre of the video image is chosen as the fovea by default. As the video session progresses, the position of the fovea will be dynamically altered to the point of interest in the video scene. The user is given the option to specify the neighbourhood around the fovea which is subjectively more important in the scene. Information within this area is always sent in HP cells. The background information is sent in two bursts: the low frequencies are sent as HP cells and the high frequencies as LP cells. The result of this scheme is a reduction of the subjective deterioration of the resultant image in the event of cell loss in the ATM network. Only the LP cells comprising the high frequency coefficients of the background information will be lost.

Thus, using this technique restricts cell loss to regions of little interest in the image. Using the technique of transform coding alone does not give the advantage of specifying a spatial area of importance in the image. Also, if the Spatially Varying Sensing concept is used all by itself, it might result in cells of the background getting completely lost. The resultant image quality will again be poor. Hence, we have used a combination of Spatially Varying Sensing and Combined Transform Coding techniques in the design of our video codec. The combination also helps to decrease the redundancy of the layered structure, which has remained one of the drawbacks in traditional layered coding algorithms. If the image has no centre of interest, we can still use Combined Transform Coding alone to good effect.

3.3 Application considerations in codec design

The requirements for compressed video on digital storage media (DSM) have a natural impact on the design of the video codec. We have implemented an application which supports playback of recorded audio and video information. The complete set of features of this application will be described in **Chapter 5**. Hence, our compression algorithm must have the capability to fulfill requirements of the orchestrated application. The following features [8] have been identified as important in order to meet the needs of our application.

Access to specific video units: In our application, a video session is recorded and stored as a set of small video playback units. Each unit represents an access point from where we can start playback of the video session. To enable this feature, we need existence of segments of information coded only with reference to themselves. Hence, the video compression algorithm must periodically send portions of a frame without reference to the previous frame. This is a kind of command refreshment scheme built into the codec to support access to specific video units. Without this command refreshment scheme, it might take an annoyingly long time for the entire video frame to be displayed.

Now, high compression demands force us to use inter-frame coding, in which only portions of the video frame which are different from the earlier frame are transmitted. There is a tradeoff at this point. Quality requirements of distributed multimedia applications demand a very high compression ratio not achievable with intra-frame coding alone. On the other hand, the access requirement to specific portions of the compressed video stream can be best satisfied by pure intra-frame coding. Our algorithm can satisfy all the requirements only insofar as it achieves the high compression ratio associated with inter-frame coding, while not compromising random access for those applications that demand it. This requires a delicate balance between intra and inter-frame coding. We have resorted to the command refreshment scheme as a solution to this problem.

Fastforward/Reverse searches: Our orchestrated application supports fastforward and reverse searches. To provide for this feature, we again need to have access to video frames coded only with reference to themselves. The reason is the same as in the previous case. Using the command refreshment scheme again helps to build an entire video frame in a short time.

Audio-Visual synchronization: We have provided techniques to facilitate audio-video synchronization in our multimedia application. These techniques entail adding appropriate timing and sequence number information to compressed video packets, which is used to time its output. Otherwise, this feature has no bearing on the functionality of the video codec as far as the compression algorithm is concerned.

Robustness to errors: Our layered coding scheme mainly addresses this point. It is a compensation technique to alleviate the problems due to random errors in the communication links and cell dropping during network congestion. The command refreshment technique also prevents the problems due to the propagative nature of VBR coding.

Coding/Decoding delay: Distributed, live, multimedia applications involving motion video need to keep the total system delay under 150 ms in order to maintain the conversational, *face-to-face* nature of the application. Since quality and delay can be traded-off to a certain extent, the algorithm should perform well over the range of acceptable delays. Our choice of Haar Wavelet Transform (HT) as against the conventional Discrete Cosine Transform (DCT) as a transform coding algorithm was based on the need to decrease the coding delay. DCT is computationally very intensive though it effects more compression than HT.

3.4 Stages in the SPAFLAY video codec

The codec has two modes of operation: *prioritized mode* and *normal mode.* The *prioritized mode* is suited to applications like teleconferencing which have a distinct area of interest. In this mode, we use a combination of layering in the spatial domain and the frequency domain. We will mainly concentrate on the functionality of the codec in this mode.

The *normal mode* of operation is suited to applications without a marked fovea. In this mode, there is no prioritization of data either in the spatial or the frequency domain. The codec uses a subset of the steps in the *prioritized mode.* We will point out the differences in this mode of operation at appropriate places.

The coding algorithm involves the following steps:

- The first step is **Block analysis**. The image is split into blocks of 8x8 pixels. For each period, a block is compared to the corresponding block in the previous image. If there is no *significant* change in the block, no information will be transmitted for this block. If the block is significantly *different*, it is subjected to further compression steps and transmitted.

- The second step is **Transform coding**. For each block which has to be transmitted, a Haar Wavelet Transform (HT) is performed. As the adjacent pixels tend to be correlated, this transformation statistically reduces the amount of information which will have to be transmitted.

- The third step is **Layering**. In the *normal mode*, this step is skipped. The Haar Transform coefficients go directly to the thresholding stage.

  The Layering stage assumes significance only in the *prioritized mode.* Here, we split the blocks to be transmitted into high priority and low priority groups on the basis of their spatial position in the video frame.

The HT coefficients of the low priority blocks are again divided into two priority groups using frequency splitting. The low frequency coefficients are thresholded, sequenced in a zig-zag fashion, RunLength encoded and transmitted as high priority data. The other group representing high frequency coefficients is also thresholded, sequenced in a zig-zag manner, RunLength encoded and transmitted as low priority data.

- The fourth step is **Thresholding**. This step achieves further compression by eliminating those Haar Wavelet coefficients which do not unduly affect the quality of the decoded image.
- The fifth step is **Command refreshment**. Here, we periodically transmit portions of the video frame irrespective of their motion content. They are also transmitted at high resolution by skipping the thresholding stage. Command refreshment improves image quality.
- The sixth step is **Zig-zag sequencing** of Haar Transform coefficients to facilitate entropy coding.
- The last step is **RunLength encoding**. Once the HT coefficients have been computed, thresholded and sequenced in a zig-zag manner, an entropy coding scheme is applied to transform the image to a string of bits.

3.4.1 Block analysis

Block analysis is used to reduce the temporal redundancy. The current frame is compared to the previous frame, looking for portions of the image that have changed significantly. Only the areas which have changed in the current frame with respect to the previous one will be retransmitted. The differencing step can be done quickly, reducing not only the bandwidth required to send video information, but also the total amount of work to compress the frame, since only the areas which change need to be processed by the subsequent compression steps. In a typical teleconference scenario, this step will achieve a compression ratio of 3:1 or more.

This step is performed after dividing the frame into blocks of size 8x8 pixels. To detect the portions of the current video frame which have changed with respect to the previous frame, we compare the luminance components of similar blocks in the two frames. If this difference between the two blocks exceeds a particular threshold, the block is marked as a *motion block* and is subjected to further compression steps before being transmitted. A unique feature of SPAFLAY is the scheme to detect areas of motion. In a teleconference scenario, it is reasonable to expect more motion in the area around the fovea. Limited motion in the background is not particularly important to the viewer. Hence SPAFLAY is more sensitive to motion around the fovea. This is achieved by having a variable threshold to detect *motion blocks*.

Accordingly, this threshold varies with the block position. If the block is closer to the fovea, it has a lower threshold. As we go away from the fovea, the threshold increases. Our experiments suggest that a threshold range (difference in the luminance components of corresponding blocks) from 50 (z_0) to 120 (z') is effective to detect motion. Let (x_0, y_0) be the block containing the fovea. The threshold value (z_0) is associated with this block. We designate the block which is most distant from the fovea block as (x', y'). The highest threshold value (z') is associated with this block. All other blocks are assigned values between z_0 and z'. A threshold value is associated with other blocks in a video frame as follows:

Consider the parabola

$$(x - x_0)^2 + (y - y_0)^2 = 4a(z - z_0) \tag{3.1}$$

where (x, y) represents a block in the video frame with a threshold value z. Substituting $(x, y) = (x', y')$ and $z = z'$ in this equation, we get the value of a:

$$a = \frac{(x' - x_0)^2 + (y' - y_0)^2}{4(z' - z_0)}$$

Substituting the value of a in equation 3.1, we get

$$(x - x_0)^2 + (y - y_0)^2 = 4\frac{(x' - x_0)^2 + (y' - y_0)^2}{4(z' - z_0)}(z - z_0) \tag{3.2}$$

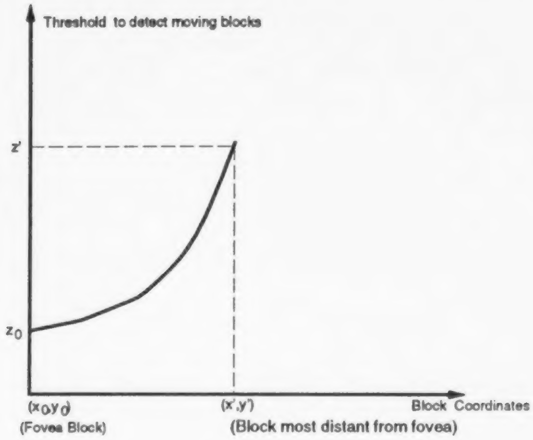


Figure 3: Threshold for motion detection

A threshold value (z) for a block is obtained by substituting (x, y) with the coordinates of the block.

The threshold values assigned to blocks in the video frame appear as shown in Fig. 3. It is clear that the threshold values closer to the fovea are small and they increase progressively as we move away from the fovea. Hence, even a little motion close to the fovea causes the block to be interpreted as a *motion block*, whereas, we need substantial motion in the background areas for the block to be classified as a *motion block*. We effect more compression this way, by restricting our choice to an area of importance. In this sense, SPAFLAY is different from other motion detection schemes which tend to use a uniform criterion to detect motion across the entire video frame.

This suits a marked fovea application like teleconferencing. We endevour to achieve through this step, the same advantage which we have in Variable Resolution algorithms [3, 4]. In these algorithms, compression is achieved by resorting to controlled sub-sampling. More pixels are sampled closer to the fovea and we progressively decrease the number of samples as we move away from the fovea. Also, Variable Resolution algorithms applied to image compression typically operate in the spatial domain. This results in constant bit-rate (CBR) traffic. As we explained earlier, ATM favours a VBR traffic. Hence, we mimic the Variable Resolution algorithms only to the extent of exercising selection over certain regions of interest to us in the video frame. We will thereafter resort to transform coding of these selected blocks producing a VBR compressed video stream.

If the application does not have a marked fovea, the user may choose to ignore it. We provide the *normal mode* of operation wherein the threshold is uniform across the entire image. In this case, the lower threshold z_0 is selected as the threshold to detect motion for all blocks in the video frame.

3.4.2 Transform coding

Each *motion block* is compressed further using transform coding. The transform is a two-dimensional Haar Wavelet. The blocks have a high spatial redundancy. The redundancy reduction techniques usable to this effect are many, but because of the block-based nature of the motion compensation process, block-based techniques are preferred. In the field of block-based spatial redundancy techniques, transform coding techniques and vector quantization are the two likely candidates. Transform coding techniques with a combination of visually weighted scalar quantization and run-length encoding have been preferred to vector quantization owing to their relatively straightforward implementation.

The Haar Transform [2, 6] is based on a class of orthogonal matrices whose elements are either 1, -1, or 0 multiplied by powers of $\sqrt{2}$. The orthonormal Haar Transform is a computationally efficient image transform. The transform of an N-point vector requires only $2(N-1)$ additions and N multiplications. The Haar Transform basis vectors for N=8 is shown in Fig. 4. The Haar matrix for N=8 is shown in Fig. 5. It is to be noted that the product of the Haar matrix with a vector results in rough coarse-to-fine sampling. The first element gives the mean value of the components. The second results in an average difference of the first 1/2N components and the second 1/2N components. The remaining elements of the product measure the adjacent differences of data elements taken four at a time or two at a time. It must

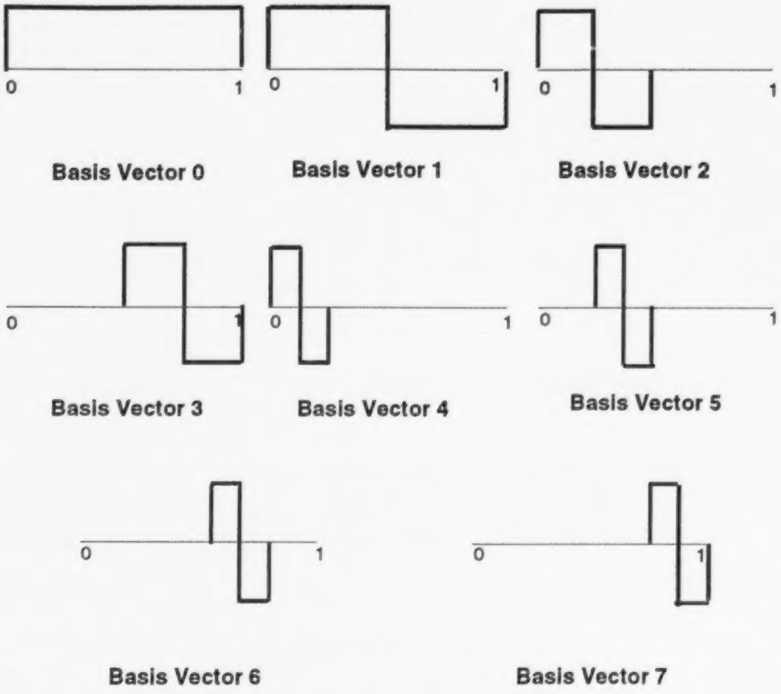

0
1
Basis Vector 0
Basis Vector 1
Basis Vector 2
Basis Vector 3
Basis Vector 4
Basis Vector 5
Basis Vector 6
Basis Vector 7

Figure 4: Haar basis vectors for N=8

be noted that the Haar Transform is locally and globally sensitive unlike transforms like Fourier which are globally sensitive.

From the basis vectors in Fig. 4, we notice that some of the basis vectors have finite values over only a small part of their range. This makes the transform locally as well as globally sensitive to image detail.

The Haar Transform of an image may be computed by

$$[F(u,v)] = [H][f(x,y)][H]^{-1}, \tag{3.3}$$

and the inverse transform is given by

$$[f(x,y)] = [H]^{-1}[F(x,y)][H], \tag{3.4}$$

where the Haar matrix is obtained by sampling the set of Haar functions. The two-dimensional basis matrices for N=8 are schematically presented in Fig. 6 [15].

Haar Transform-based compression is essentially a compression of 8x8 blocks of grayscale image samples. Color image compression can be approximately regarded as

$$H=1/\sqrt{8}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{bmatrix}$$

Figure 5: Orthonormal Haar matrix for N=8

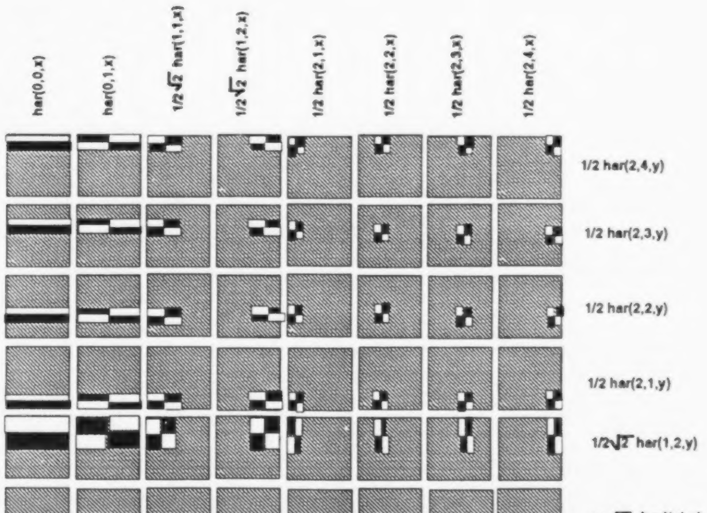

har(0,0,x)
har(0,1,x)
1/2√2 har(1,1,x)
1/2√2 har(1,2,x)
1/2 har(2,1,x)
1/2 har(2,2,x)
1/2 har(2,3,x)
1/2 har(2,4,x)
1/2 har(2,4,y)
1/2 har(2,3,y)
1/2 har(2,2,y)
1/2 har(2,1,y)
1/2√2 har(1,2,y)
1/2√2 har(1,1,y)
har(0,1,y)
har(0,0,y)

Figure 6: Two-dimensional Haar basis matrices for N=8

compression of multiple grayscale images, which are either compressed entirely one at a time, or are compressed by alternately interleaving 8x8 sample blocks from each in turn. For color images, we subject the luminance and chrominance components of a block separately to Haar Transform. Input to the Forward Haar Transform is an 8x8 sample block. Each 8x8 block of source image samples is effectively a 64-point discrete signal which is a function of the two spatial dimensions x and y. The Forward Haar Transform takes this signal as its input and decomposes it into 64 Haar Transform coefficients. The coefficient with zero frequency in both dimensions is called *DC coefficient* and the remaining 63 coefficients are called *AC coefficients.* Because the sample values typically vary slowly from point to point across an image, the Forward Haar Transform processing step lays the foundation for achieving data compression by concentrating most of the signal in the lower spatial frequencies. We get a compression ratio of 6:1 or more as a result of transform coding followed by entropy coding.

At the decoder, the Inverse Haar Transform reverses this processing step. It takes the 64 Haar Transform coefficients (which at that point have been thresholded) and reconstructs a 64-point output image signal.

We associate a *Quality-Index* (QI) parameter with each of the 64 Haar Transform coefficients. This index is assigned as follows: First, the Haar basis vector 2 and the Haar basis vector 3 (Fig. 4) are treated as a single unit. Similarly, basis vectors 4, 5, 6 and 7 are treated as one unit. The other vectors are considered as separate units. This step is necessitated to group together vectors which are only locally significant. With this step, we can consider the 8x8 two-dimensional basis matrix (Fig. 6) as a 4x4 matrix of basis vector units. This is shown in Fig. 7. Each HT coefficient corresponding to an element in the 4x4 basis matrix is assigned a *Quality-Index* which is equal to the sum of the row and the column number of the element in the 4x4 matrix. Effectively, we have labelled each of the 64 HT coefficients corresponding to elements in the 8x8 basis matrix with a *Quality-Index* as in Fig. 7. The *Quality-Index* of a Haar Transform coefficient gives a measure of the image detail carried by it. A

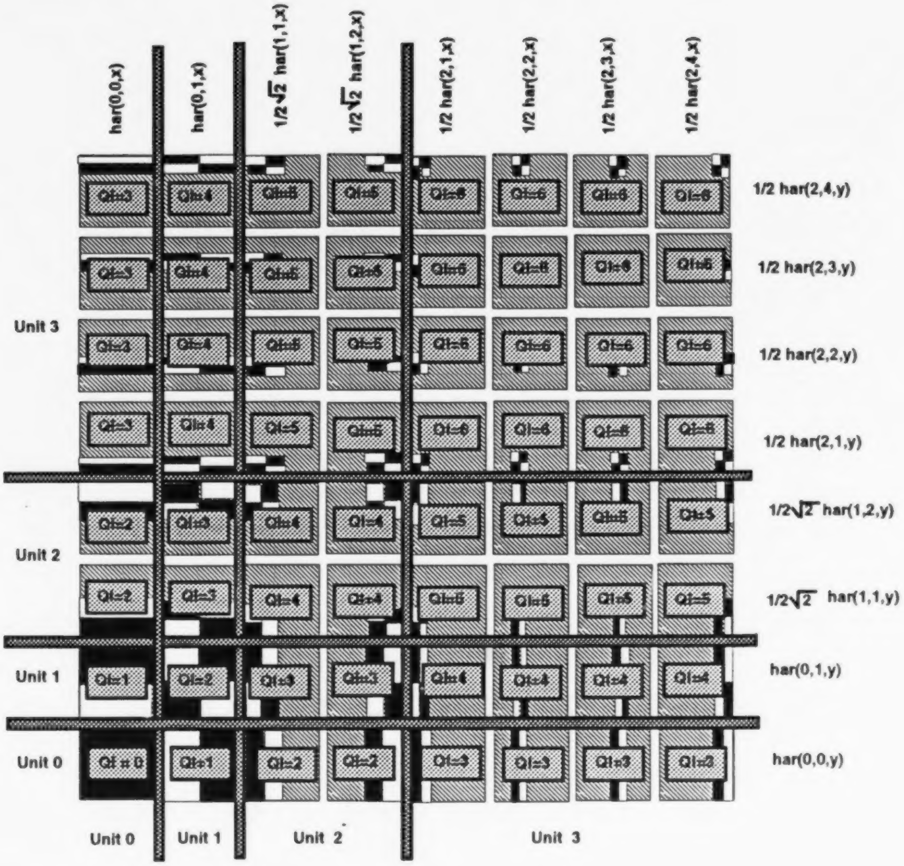

har(0,0,x)
har(0,1,x)
1/2 √2 har(1,1,x)
1/2 √2 har(1,2,x)
1/2 har(2,1,x)
1/2 har(2,2,x)
1/2 har(2,3,x)
1/2 har(2,4,x)
1/2 har(2,4,y)
1/2 har(2,3,y)
1/2 har(2,2,y)
1/2 har(2,1,y)
1/2√2 har(1,2,y)
1/2√2 har(1,1,y)
har(0,1,y)
har(0,0,y)
Unit 3
Unit 2
Unit 1
Unit 0
QI=3 QI=4 QI=5 QI=5 QI=6 QI=6 QI=6 QI=6
QI=3 QI=4 QI=5 QI=5 QI=6 QI=6 QI=6 QI=6
QI=3 QI=4 QI=5 QI=5 QI=6 QI=6 QI=6 QI=6
QI=3 QI=4 QI=5 QI=5 QI=6 QI=6 QI=6 QI=6
QI=2 QI=3 QI=4 QI=4 QI=5 QI=5 QI=5 QI=5
QI=2 QI=3 QI=4 QI=4 QI=5 QI=5 QI=5 QI=5
QI=1 QI=2 QI=3 QI=3 QI=4 QI=4 QI=4 QI=4
QI = 0 QI=1 QI=2 QI=2 QI=3 QI=3 QI=3 QI=3
Unit 0
Unit 1
Unit 2
Unit 3

Figure 7: Quality-Index (QI) values for Haar Transform coefficients

Haar Transform coefficient with a low index value indicates coarse sampling with finer details being progressively added by coefficients with higher *Quality-Index* values.

3.4.3 Layering in SPAFLAY

This step is significant only in the *prioritized mode* of operation of the codec. It is skipped in the *normal mode* operation. In the *normal mode*, the transform coded blocks are directly subjected to thresholding.

The ATM cell header consists of the following fields: generic flow control (GFC), virtual path identifier (VPI), virtual channel identifier (VCI), payload type (PT), cell loss priority (CLP) and header error control (HEC). The header format is different at

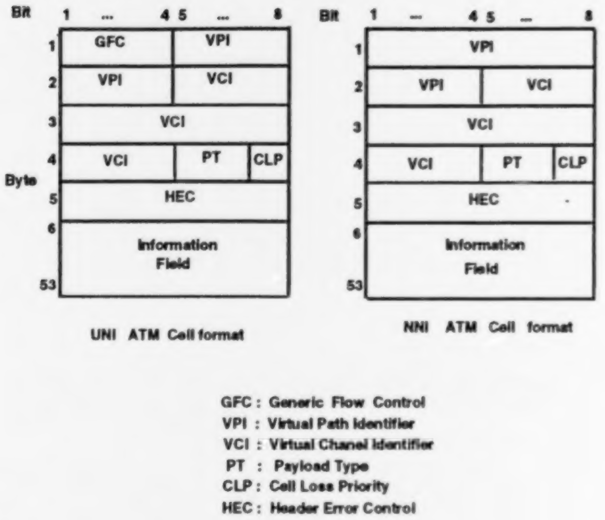

Bit
1 ... 4 5 ... 8
Byte
1
2
3
4
5
6
53
GFC
VPI
VPI
VCI
VCI
VCI
PT
CLP
HEC
Information Field
UNI ATM Cell format
Bit
1 ... 4 5 ... 8
1
2
3
4
5
6
53
VPI
VPI
VCI
VCI
VCI
PT
CLP
HEC
Information Field
NNI ATM Cell format
GFC : Generic Flow Control
VPI : Virtual Path Identifier
VCI : Virtual Chanel Identifier
PT : Payload Type
CLP : Cell Loss Priority
HEC : Header Error Control

Figure 8: ATM Cell

a B-ISDN user network interface (UNI) than it is in a B-ISDN network node interface (NNI), as illustrated in Fig. 8.

The CLP field of the ATM cell header is a 1-bit field used for cell-loss priority. Due to the statistical multiplexing of connections, it is unavoidable that cell losses will occur in a B-ISDN. A cell with CLP bit set may be discarded by the network during congestion, whereas cells with the CLP bit cleared, have a higher priority and shall not be discarded if at all possible.

The CLP bit present in the ATM cell header is used to have two levels of priority in an ATM network. Our coding strategy is based on a judicious usage of this field to send prioritized information over the ATM network. A high priority is assigned to those elements of a compressed video frame which will increase the subjective quality of the decoded image. The choice of priority should be based on the criterion that even if the low priority elements are completely lost in the network, it will not have a bearing on the decoding of the video image at the receiving end. Also, the high priority elements of the video frame alone should be able to guarantee a decoded image of *acceptable* quality.

Accordingly, there are two ways of assigning priority in SPAFLAY: *Spatial layering*

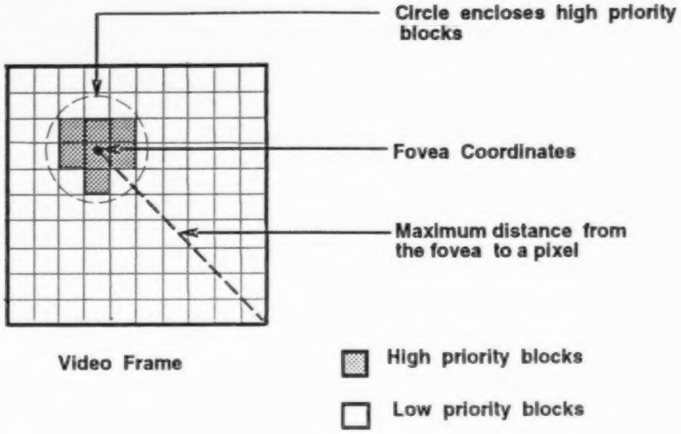

Circle encloses high priority blocks
Fovea Coordinates
Maximum distance from the fovea to a pixel
Video Frame
High priority blocks
Low priority blocks

Figure 9: Spatial prioritization of blocks

and *Frequency layering*.

Spatial layering: Our first task is to assign priority to portions of the video frame on the basis of their spatial position. As we have stated earlier, the *prioritized mode* of operation caters to applications with a marked fovea. In these applications, the area around the fovea will be subjectively more important to the viewer than the rest of the picture.

At the start of the video session, the centre of the image is chosen as the fovea. The user also specifies the value for the *fovea-neighbourhood* parameter. It is input as a percentage. We determine the maximum distance which a pixel may have from the fovea. Then, a circle with the fovea as the centre and *fovea-neighbourhood* percentage of this maximum distance as the radius, determines the high priority region. The blocks of the video frame which lie within this circle are designated as high priority blocks and the blocks which lie outside this circle are low priority blocks. This process is illustrated in Fig. 9.

When one of these high priority blocks is marked as a *motion block* in the **Block analysis** step, it will pass through the entire compression procedure. This involves performing a Forward Haar Wavelet Transform over the block. The resultant Haar Wavelet Transform coefficients are thresholded, sequenced in a zig-zag fashion, RunLength encoded and then transmitted in ATM cells with the CLP bit set to zero, signifying high priority data. This is layered coding as applied to video frame blocks on the basis of their spatial location.

We begin the video session with the fovea fixed at the centre of the image. Thereafter, the position of the fovea will be dynamically altered. It is natural to associate the fovea with the point of maximum motion in the video scene. This fact is used to dynamically determine the fovea position in the video image. The **Block Analysis** step keeps an account of the number of times each video frame block is transmitted for a given time interval. At the end of the time interval, the block which was transmitted maximum number of times is designated as the *fovea block*. The pixel at the centre of the *fovea block* becomes the new fovea. We recompute the low priority and high priority labels assigned to video blocks with respect to this new fovea. Currently, this computation to determine the new fovea position is done every 15 frames.

Frequency layering: The blocks which fall outside the selected neighbourhood around the fovea are low priority blocks. Over these low priority blocks, we use another technique of layering which is based on splitting of frequencies. This technique is also known as **Combined Transform Coding**. It is based on the rationale that using just the low frequency components from the transform coded block, we will be able to reconstruct the entire block at the decoder end. The quality of the decoded block will be low. Presence of high frequencies enhances the decoded block quality since these frequencies account for the sharpness and details of an image. However, it should be clearly noted that low frequencies alone are adequate to decode the entire block, though the resultant block will be of inferior quality.

Our goal is to send the low frequencies of a block as high priority data over the ATM network by packing them in ATM cells with the CLP bit set to zero. The high frequencies of the same block will be sent as low priority data by setting the CLP bit to one in all the cells carrying this data. We will interleave these high priority and low priority cells while sending them over the ATM network. At the decoder end, we merge the high and low frequencies of the block and then decode the resultant block. If the high frequencies (low priority data for the network) are lost, the block will be decoded using only the low frequency components of the block.

Here, we need to quantify the values of our *low frequencies* and *high frequencies.* We require a threshold such that all frequencies below the threshold will be marked as *low frequencies* and frequencies above it will be *high frequencies.* As we increase the threshold, more frequencies get classified as *low frequencies.* This will increase the quality of the decoded block, but it will also increase the volume of data sent as high priority over the network, which might result in increased network congestion. If the threshold is fixed very low, then we increase the volume of data sent as low priority over the network. This raises the possibility of more cells getting dropped by the network since they were assigned a low priority. The result might be a decoded block of poor quality. Hence, the threshold to split *low frequencies* and *high frequencies* must be carefully chosen by taking into account the acceptable quality level in our application and the anticipated state of the network.

We allow the user to specify this threshold as a **Quality** parameter at the start of the video session. It has a range from 0 to 6. As we saw earlier, the input to the Forward Haar Transform is an 8x8 pixel block. The output is an 8x8 block of Haar Transform coefficients. The *Quality-Index* parameter assigned to each Haar Transform coefficient will be used in our layered coding approach. The layered coding now works as follows:

A low priority motion block (designated on the basis of its position in the video frame) is subjected to the Forward Haar Transform. The output 8x8 element block of Haar Transform Coefficients is called *HTBase-block.* We save a copy of *HTBase-block* in *HTCopy-block.* The algorithm starts by considering *HTBase-*

block.

In *HTBase-block*, the *Quality-Index* of each HT coefficient is compared with the input **Quality** parameter. If the *Quality-Index* of the HT coefficient is less than or equal to the **Quality** parameter, then the HT coefficient is made zero. A little analysis will clarify the fact that *HTBase-block* after the above procedure, represents a block with its low frequencies set to zero. Only the *high frequencies* of the original HT block are nonzero. The block is then thresholded, sequenced in a zig-zag fashion, RunLength encoded and packed into ATM cells with the CLP bit set to one and then transmitted. With this, we have sent the high frequencies of the original HT block as low priority data.

Next, we consider the original HT block which had been saved in *HTCopy-block*. In *HTCopy-block*, the *Quality-Index* of each HT coefficient is compared with the input **Quality** value. If the *Quality-Index* of each HT coefficient is greater than the **Quality** parameter, then the HT coefficient is made zero. *HTCopy-block* after the above computation, represents a block with its high frequencies set to zero. The *low frequencies* of the original HT block are alone nonzero. The resultant block is again thresholded, sequenced in a zig-zag manner, RunLength encoded and packed into ATM cells with the CLP bit set to zero and then transmitted. With this, we have sent the low frequencies of the original HT block as high priority data.

Thus, the HT coefficients of a block are transmitted in two bursts. In the first burst, the high frequencies of the block are transmitted as low priority data over the ATM network. In the second burst, the low frequencies of the block are sent as high priority data over the ATM network. In effect, we have divided the HT coefficients in an HT block into two groups of high priority and low priority coefficients on the basis of the value of the **Quality** parameter. The high priority and low priority HT coefficients are thereafter transmitted separately over the network.

Quality	No. of HP coeffs.	No. of LP coeffs.
0	1	63
1	3	61
2	8	56
3	20	44
4	32	32
5	48	16
6	64	0

Table 3.1: Breakup of high and low priority HT coefficients

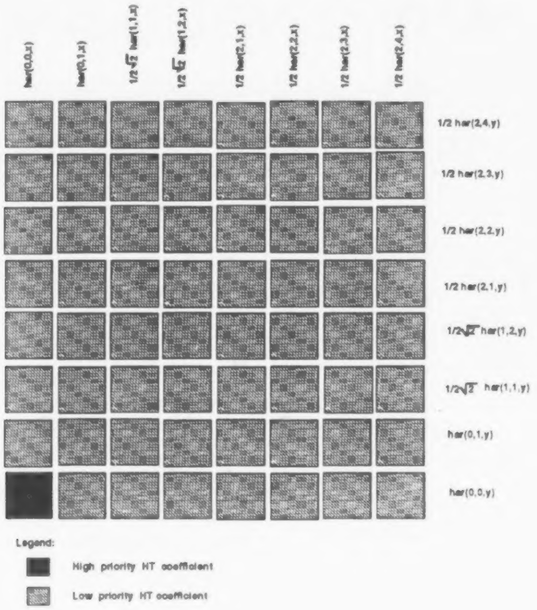

har(0,0,x)
har(0,1,x)
1/2 √2 har(1,1,x)
1/2 √2 har(1,2,x)
1/2 har(2,1,x)
1/2 har(2,2,x)
1/2 har(2,3,x)
1/2 har(2,4,x)
1/2 har(2,4,y)
1/2 har(2,3,y)
1/2 har(2,2,y)
1/2 har(2,1,y)
1/2√2 har(1,2,y)
1/2√2 har(1,1,y)
har(0,1,y)
har(0,0,y)
Legend:
High priority HT coefficient
Low priority HT coefficient

Figure 10: Quality=0: HP and LP Haar Transform coefficients

Fig. 10 to 16 indicate the high priority and low priority HT coefficients in a given HT block for different values of the **Quality** parameter. It is seen that for a **Quality** value of 0, one HT coefficient is sent as high priority data and the other 63 coefficients of the HT block are sent as low priority data. When the **Quality** value is 6, all the 64 HT coefficients in the HT block are sent as high priority data. The breakup of the number of high priority and low priority HT coefficients in an HT block for different values of the **Quality** parameter are summarised in **Table 3.1**.

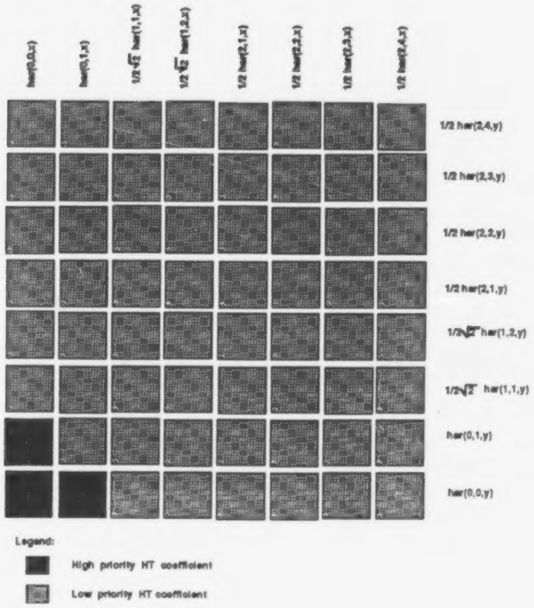

har(0,0,x)
har(0,1,x)
1/2√2 har(1,1,x)
1/2√2 har(1,2,x)
1/2 har(2,1,x)
1/2 har(2,2,x)
1/2 har(2,3,x)
1/2 har(2,4,x)
1/2 har(2,4,y)
1/2 har(2,3,y)
1/2 har(2,2,y)
1/2 har(2,1,y)
1/2√2 har(1,2,y)
1/2√2 har(1,1,y)
har(0,1,y)
har(0,0,y)
Legend:
High priority HT coefficient
Low priority HT coefficient

Figure 11: Quality=1: HP and LP Haar Transform coefficients

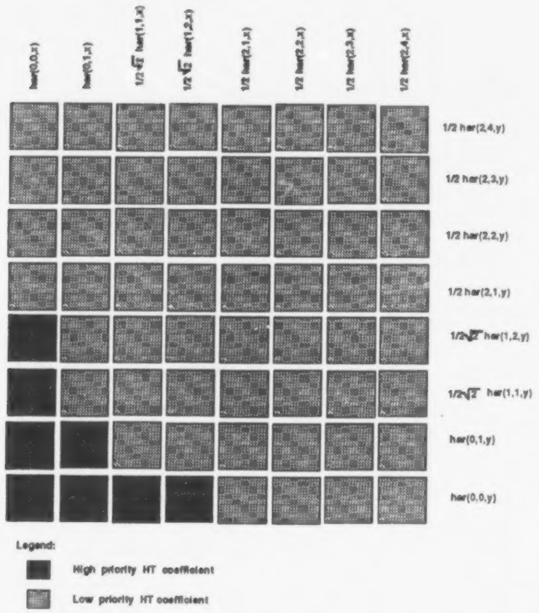

har(0,0,x)
har(0,1,x)
1/2√2 har(1,1,x)
1/2√2 har(1,2,x)
1/2 har(2,1,x)
1/2 har(2,2,x)
1/2 har(2,3,x)
1/2 har(2,4,x)
1/2 har(2,4,y)
1/2 har(2,3,y)
1/2 har(2,2,y)
1/2 har(2,1,y)
1/2√2 har(1,2,y)
1/2√2 har(1,1,y)
har(0,1,y)
har(0,0,y)
Legend:
High priority HT coefficient
Low priority HT coefficient

Figure 12: Quality=2: HP and LP Haar Transform coefficients

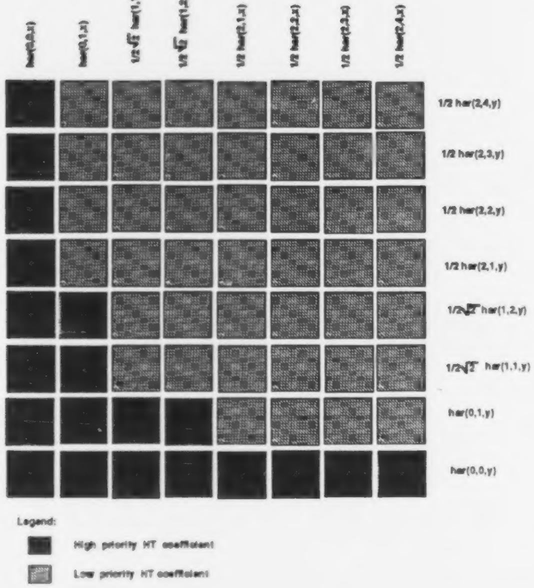

har(0,0,x)
har(0,1,x)
1/2 √2 har(1,1,x)
1/2 √2 har(1,2,x)
1/2 har(2,1,x)
1/2 har(2,2,x)
1/2 har(2,3,x)
1/2 har(2,4,x)
1/2 har(2,4,y)
1/2 har(2,3,y)
1/2 har(2,2,y)
1/2 har(2,1,y)
1/2√2 har(1,2,y)
1/2√2 har(1,1,y)
har(0,1,y)
har(0,0,y)
Legend:
High priority HT coefficient
Low priority HT coefficient

Figure 13: Quality=3: HP and LP Haar Transform coefficients

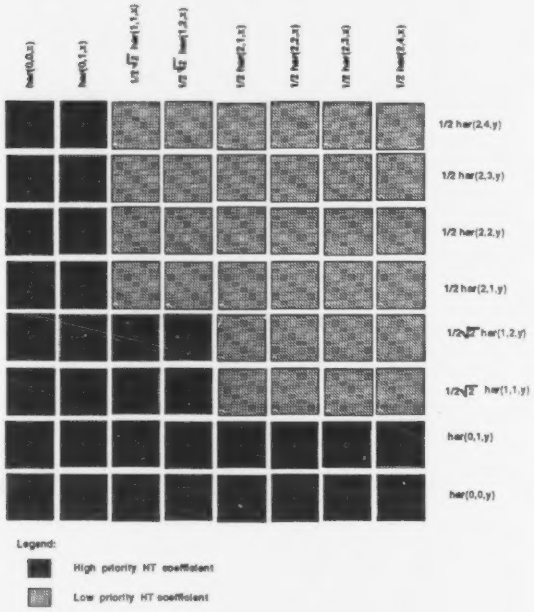

har(0,0,x)
har(0,1,x)
1/2 √2 har(1,1,x)
1/2 √2 har(1,2,x)
1/2 har(2,1,x)
1/2 har(2,2,x)
1/2 har(2,3,x)
1/2 har(2,4,x)
1/2 har(2,4,y)
1/2 har(2,3,y)
1/2 har(2,2,y)
1/2 har(2,1,y)
1/2√2 har(1,2,y)
1/2√2 har(1,1,y)
har(0,1,y)
har(0,0,y)
Legend:
High priority HT coefficient
Low priority HT coefficient

Figure 14: Quality=4: HP and LP Haar Transform coefficients

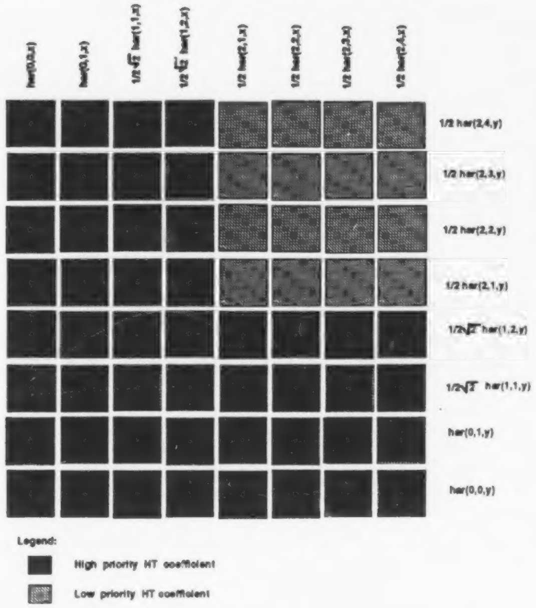

har(0,0,x)
har(0,1,x)
1/2√2 har(1,1,x)
1/2√2 har(1,2,x)
1/2 har(2,1,x)
1/2 har(2,2,x)
1/2 har(2,3,x)
1/2 har(2,4,x)
1/2 har(2,4,y)
1/2 har(2,3,y)
1/2 har(2,2,y)
1/2 har(2,1,y)
1/2√2 har(1,2,y)
1/2√2 har(1,1,y)
har(0,1,y)
har(0,0,y)
Legend:
High priority HT coefficient
Low priority HT coefficient

Figure 15: Quality=5: HP and LP Haar Transform coefficients

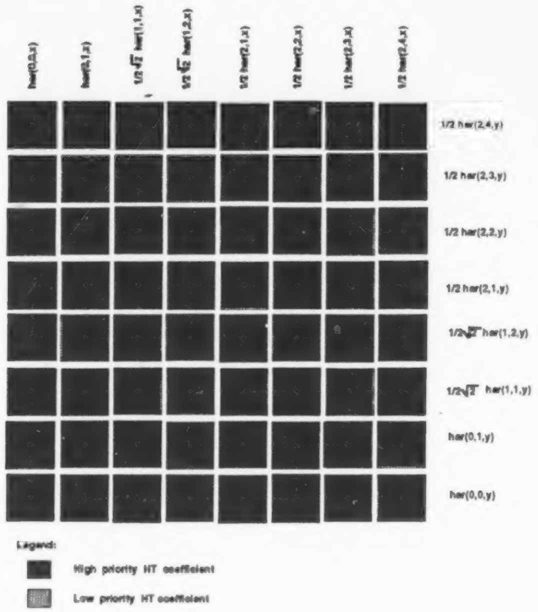

har(0,0,x)
har(0,1,x)
1/2√2 har(1,1,x)
1/2√2 har(1,2,x)
1/2 har(2,1,x)
1/2 har(2,2,x)
1/2 har(2,3,x)
1/2 har(2,4,x)
1/2 har(2,4,y)
1/2 har(2,3,y)
1/2 har(2,2,y)
1/2 har(2,1,y)
1/2√2 har(1,2,y)
1/2√2 har(1,1,y)
har(0,1,y)
har(0,0,y)
Legend:
High priority HT coefficient
Low priority HT coefficient

Figure 16: Quality=6: HP and LP Haar Transform coefficients

At the decoder, we try to reconstruct the HT block by merging the low frequency components and the high frequency components. The resultant block is subjected to a Reverse Haar Transform procedure. If the high frequency components were lost due to network congestion, the low frequency components alone go through a Reverse Haar Transform procedure. In this case, the decoded block will be of relatively poor quality.

3.4.4 Thresholding

The input to this step is a block of 64 HT coefficients. The purpose of thresholding is to achieve further compression by representing HT coefficients with no greater precision than is necessary to achieve the desired image quality. The goal of this step is to discard information that is not visually significant. We have chosen a threshold value on the basis of psychovisual experiments. The low energy wavelet terms are suppressed in this step. From our experiments, it was found that HT coefficient values between -2 and 2 can be discarded without any significant loss in the decoded image quality.

3.4.5 Command refreshment

To improve the image quality and to prevent errors in VBR coding from propagating, SPAFLAY uses a form of command refreshment. Portions of the frame are transmitted irrespective of their motion content. These are blocks of the frame which are detected as *stationary blocks* over many consecutive video frames. These blocks are transform coded. But they are not thresholded. They are thus sent in *high resolution*. This process results in a nearly lossless version of the original image, while still achieving approximately 2:1 compression. With command refreshment, all the stationary portions of the image get updated to high resolution quickly after a scene change.

Command refreshment was introduced to improve the quality of multicast applications and orchestrated applications. In our application, we enable multicasting of audio and video data to a group of stations. If a client joins this multicast session while it is in progress, it will have to wait for a considerable amount of time for the entire video frame to be filled. This is due to the fact that *motion blocks* alone are retransmitted as per the compression algorithm discussed above. The video quality for the new client will be poor until the entire video frame is filled. With command refreshment, the video frame for the new client will get filled faster due to the fact that stationary portions of the video frame are also being transmitted periodically.

Another feature in our application is provision of reverse and fastforward search during playback of stored data. Moving to a far removed point in time through a rewind or fastforward can result in a complete scene change. Absence of *history* for the current video frame will result in most parts of the video frame being *void*. Absence of command refreshment can lead to very slow video frame buildup which can be annoying to the user. Command refreshment speeds up the complete frame buildup.

3.4.6 Zig-zag sequencing

The thresholded coefficients are ordered in a zig-zag sequence, as shown in Fig. 17. This ordering helps to facilitate entropy coding techniques like RunLength encoding, by placing low frequency coefficients (which are more likely to be nonzero) before high frequency coefficients. It is to be noted that the zig-zag sequence is arranged such that all the HT coefficients of a given *Quality-Index* value are packed and sent together. Also, these HT coefficients are sent in the increasing order of the *Quality-Index* values. This ordering is particularly beneficial in the *prioritized mode* of operation where entire groups of HT coefficients of a particular *Quality-Index* value are reduced to zeros.

3.4.7 Symbol-level RunLength encoding

The HT coefficients are finally subjected to RunLength encoding before being transmitted. In this step, each nonzero coefficient is represented in combination with

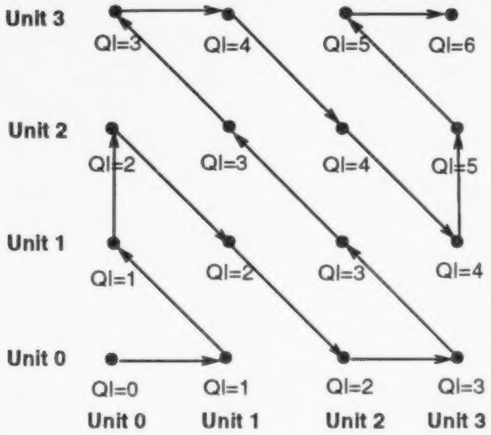


Figure 17: Zig-zag sequencing of Haar Transform coefficients

the "runlength"(consecutive number) of zero-valued coefficients which follow it in the zig-zag sequence. Each such runlength/nonzero coefficient combination is usually represented by a pair of symbols: (*symbol1, symbol2*). Symbol1 represents the runlength while symbol2 represents the "amplitude", which is simply the value of the nonzero coefficient. The runlength encoded samples are transmitted over the network.

3.4.8 Motion detection in SPAFLAY

In the **Block analysis** step, blocks of the video frame are compared with the corresponding blocks of the previous frame to detect the areas of motion in the present video frame. In this decision phase, the previous frame serves as a reference for the current frame to detect the areas of motion. Here, it is important to note that the reference frame should not be a *captured frame*. During the compression process, irreversible errors are introduced in the *captured frame* due to the lossy nature of the thresholding process. Hence, the decoded frame at the receiver's end will not be of the same quality as the *captured frame*. This decoded frame should serve as the reference frame. If the previous *captured frame* is used as the reference frame during the **Block analysis** step, then we accumulate errors due to a faulty reference frame. Hence, we need a **reconstructor** in our codec to provide the correct reference frame

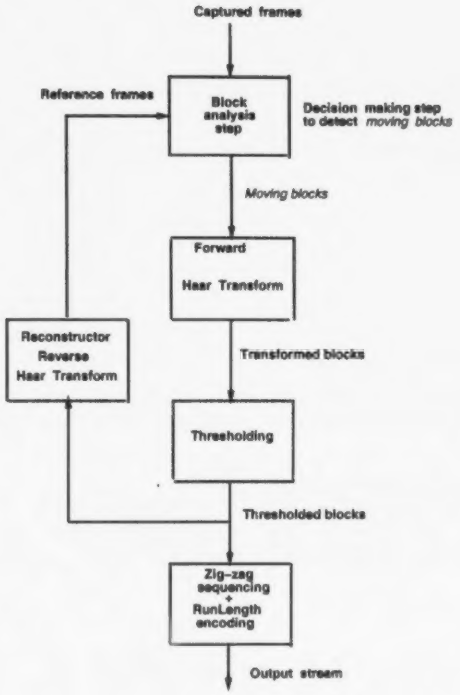

Captured frames
Reference frames
Block analysis step
Decision making step to detect *moving blocks*
Moving blocks
Forward Haar Transform
Reconstructor Reverse Haar Transform
Transformed blocks
Thresholding
Thresholded blocks
Zig-zag sequencing + RunLength encoding
Output stream

Figure 18: Reconstructor in SPAFLAY (*normal mode*)

in the decision making phase. The placement of this **reconstructor** in SPAFLAY is shown in Fig. 18.

In the *prioritized mode* of operation, the codec can be made more robust by using high priority data alone to construct a reference frame. This step ensures that motion detection is performed taking into account the information loss that might be incurred when the ATM network is congested. This robustness feature is at the potential expense of prediction efficiency in the **Block analysis** step. However, removal of some high frequency coefficients from the reference frame may not seriously impact prediction efficiency since they tend to have high variances and low inter-frame correlation. Fig. 19 shows the resulting encoder structure for the *prioritized mode* of operation.

The prediction efficiency in the **Block analysis** step can be increased by having an area of spatial importance in the image. The blocks in the neighbourhood of the fovea do not go through the frequency splitting step. Hence, the decoded quality

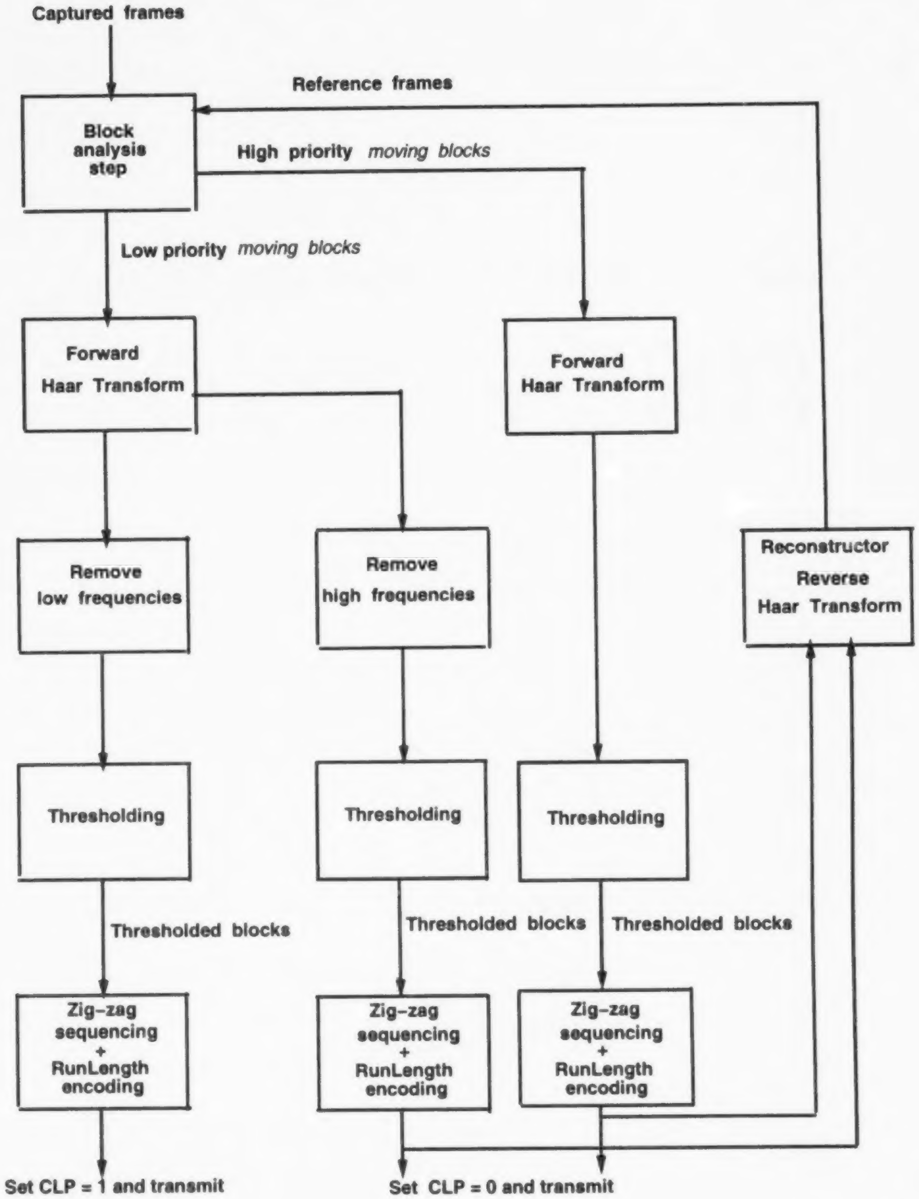

Captured frames
Reference frames
Block analysis step
High priority moving blocks
Low priority moving blocks
Forward Haar Transform
Forward Haar Transform
Remove low frequencies
Remove high frequencies
Reconstructor Reverse Haar Transform
Thresholding
Thresholding
Thresholding
Thresholded blocks
Thresholded blocks
Thresholded blocks
Zig-zag sequencing + RunLength encoding
Zig-zag sequencing + RunLength encoding
Zig-zag sequencing + RunLength encoding
Set CLP = 1 and transmit
Set CLP = 0 and transmit

Figure 19: Reconstructor in SPAFLAY (*prioritized mode*)

of these video blocks will be high. This prevents these blocks from being spuriously detected as *motion blocks* due to the coarseness of the reference frame comprising these blocks.

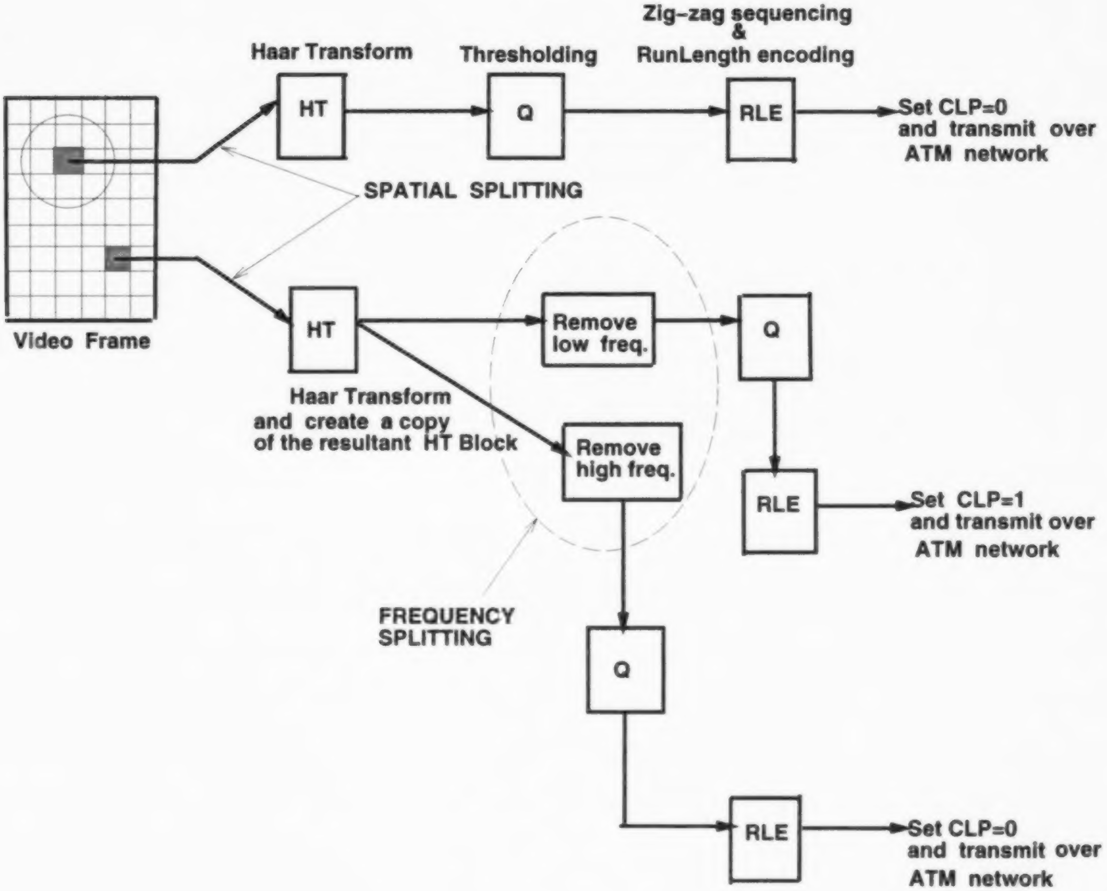
3.5 Modes of operation in SPAFLAY

3.5.1 *Prioritized mode*

The stages in the *prioritized mode* of operation of the codec are depicted in Fig. 20. It is summarised as follows:

- At the start of the video session, the centre of the video image is chosen as the fovea. As the video session progresses, the position of the fovea will be dynamically altered as discussed earlier. The user specifies the neighbourhood around the fovea which is subjectively more important. Using this information, the blocks in the video frame are classified as high priority blocks and low priority blocks. Each block also has a unique threshold value which is used to detect motion between corresponding blocks of successive video frames. In the first stage, we compare corresponding blocks of successive video frames to detect *motion blocks.* These blocks could either be high priority or low priority blocks. They are sent to the next stage.

- If the input block is a high priority block, it is subjected to Haar Transform, thresholded, sequenced in a zig-zag fashion, RunLength encoded and transmitted over the network as high priority cells. If the input block is a low priority block, it is subjected to frequency splitting. The low frequency components are thresholded, sequenced in a zig-zag fashion, RunLength encoded and transmitted over the network as high priority cells. The high frequency components also go through the same compression steps, but are transmitted as low priority cells over the ATM network.


Zig-zag sequencing
&
RunLength encoding
Haar Transform
Thresholding
HT
Q
RLE
Set CLP=0
and transmit over
ATM network
SPATIAL SPLITTING
Video Frame
HT
Remove
low freq.
Q
Haar Transform
and create a copy
of the resultant HT Block
Remove
high freq.
RLE
Set CLP=1
and transmit over
ATM network
FREQUENCY
SPLITTING
Q
RLE
Set CLP=0
and transmit over
ATM network

Legend:

▨ **Blocks detected as moving in Block analysis step**

The circle in video frame represents the neighbourhood around the fovea which is subjectively more important to the viewer

HT : Haar Transform step

Q : Thresholding step

RLE : Zig-zag sequencing + RunLength encoding step

Figure 20: *Prioritized mode* of SPAFLAY

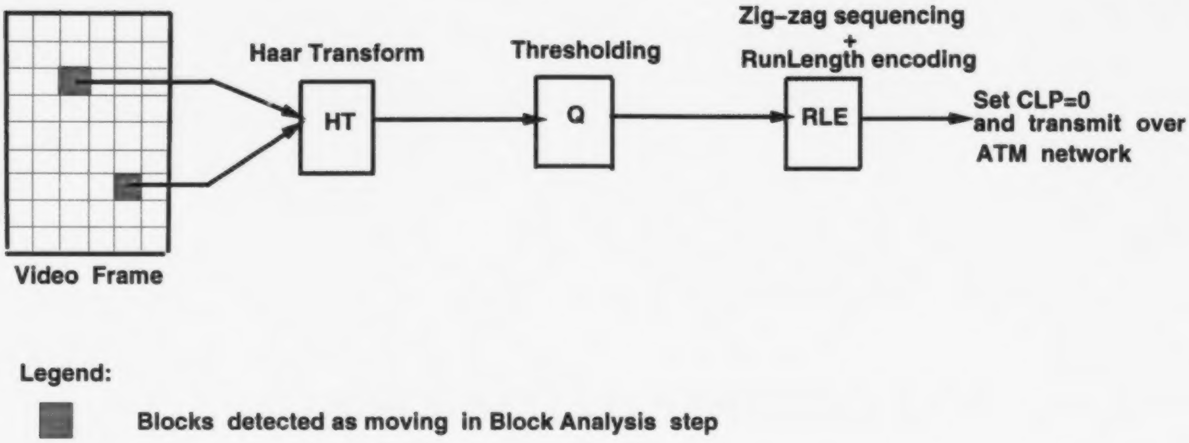

Zig-zag sequencing
+
RunLength encoding
Haar Transform
Thresholding
HT
Q
RLE
Set CLP=0
and transmit over
ATM network
Video Frame
Legend:
Blocks detected as moving in Block Analysis step

Figure 21: *Normal mode* of SPAFLAY

- Only the high priority information is used to decode the frame at the encoding end. This frame is used as the reference to detect motion for the next video frame.

- At the decoding end, the low priority block is reconstructed by merging the high and low frequencies which were sent in separate bursts over the network. The resultant block is subjected to Reverse Haar Transform and displayed.

3.5.2 *Normal mode*

The steps in the *normal mode* of operation of the codec is shown in Fig. 21. The steps are summarised as follows:

- The video scene does not have a marked fovea. Hence, all the blocks of the video frame are treated uniformly. Each block has the same threshold value which is used to detect motion between corresponding blocks of successive video frames. In the first stage, the corresponding blocks of consecutive video frames are compared to detect *motion blocks.* These blocks are sent to the next stage.

- Here, the blocks are transform coded using Haar Transform. The transform coded blocks are thresholded, sequenced in a zig-zag manner, RunLength encoded and transmitted over the network. The information transmitted over the network has a uniform priority unlike in the *prioritized mode.*

- Since there is no prioritized information in this mode, the transmitted frame is decoded at the encoding end. It serves as the reference to detect motion for the subsequent video frame.
- At the decoding end, the transmitted blocks are subjected to Reverse Haar Transform and displayed.

3.6 Use of SPAFLAY in a multicast scenario

In a video conferencing application, each conference participant sends audio and video data to other members. All the participants need not be in the same area. We can consider a case where participants in one area are connected through a low-speed link to the majority of the conference participants, who enjoy high-speed network access. Instead of forcing everyone to use lower-bandwidth, reduced quality video encoding, we could use the SPAFLAY codec to good effect.

A *router* placed near the low bandwidth area can pass only the high priority data sent by SPAFLAY to these group members. Thus we not only reduce the bandwidth to cater to these group members but also keep the quality of the image at permissible limits. This is due to the inbuilt partitioning scheme used in SPAFLAY. SPAFLAY's high priority data stream is alone sufficient to guarantee an image of reasonable quality.

Both the high priority and low priority stream produced by SPAFLAY are delivered to group members who have high-speed network access. Hence, the video image quality for these group members is superior. As a result, the members who are connected by a high-speed network are not inconvenienced due to the presence of group members connected to a low-speed network.

Thus, the natural partitioning scheme used in the SPAFLAY codec can be used to good effect in a multicast environment. This case is illustrated in Fig. 22.

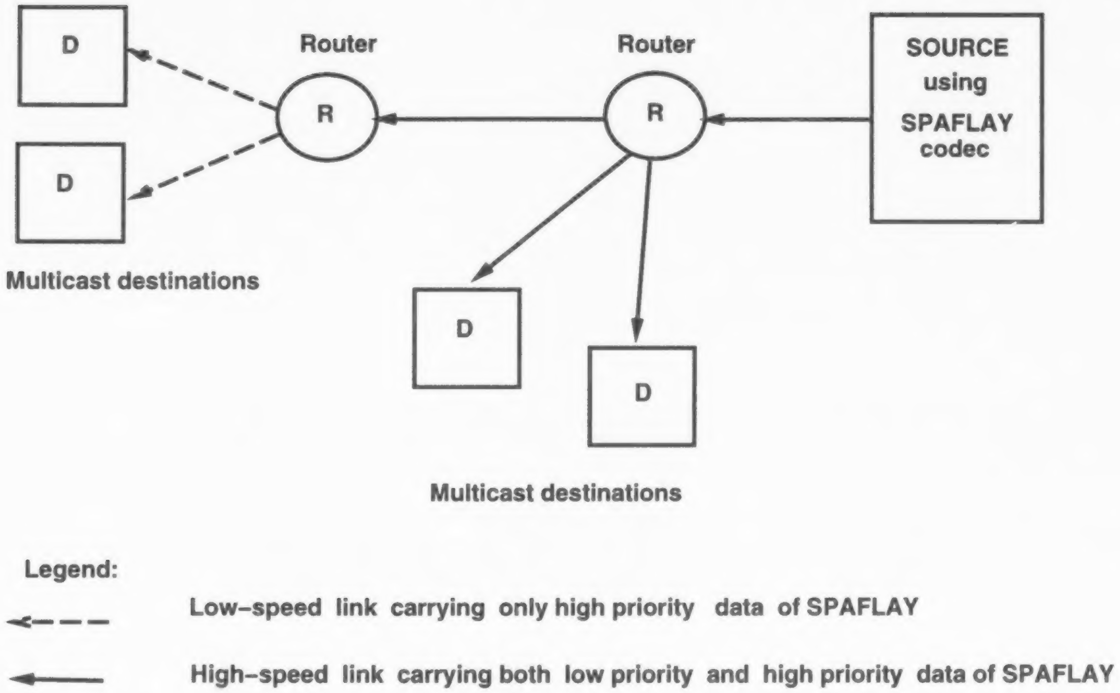

D
D
Router
R
Router
R
SOURCE using SPAFLAY codec
Multicast destinations
D
D
Multicast destinations
Legend:
Low-speed link carrying only high priority data of SPAFLAY
High-speed link carrying both low priority and high priority data of SPAFLAY

Figure 22: SPAFLAY in a multicast environment

3.7 Search in parameter space for SPAFLAY

In this section, we compress an image using the various options in SPAFLAY. In each case, we look at the decoded image quality when all the low priority cells carrying the compressed image are lost in the network. The image is reconstructed using data in the high priority cells alone.

Image quality in *normal mode* of operation (Case 1): The decoded image quality of a video scene is shown in Fig 23. In this case, the size of the compressed image was 6007 bytes. Assuming that the ATM cell payload is 48 bytes, it will take 126 ATM cells to carry the compressed video image. It must be noted that Fig. 23 depicts the quality of the image when there is no cell loss in the network. In the event of cell loss, every part of the video image is equally susceptible to loss. The worst case is when the *end-of-frame-marker* is lost in the network due to network congestion. It results in the decoder being unable to

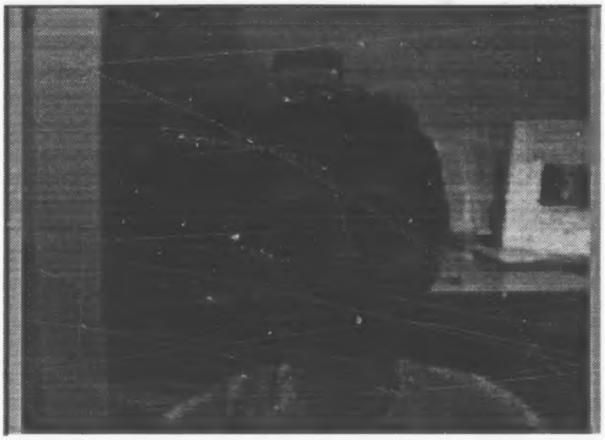


Figure 23: Image quality using *normal mode*

reconstruct the entire frame. The frame is as good as lost. Hence, the codec has least robustness when it works in the *normal mode* of operation.

Image quality in frequency layered mode (Case 2): In the prioritized mode of operation, the user can input a **Quality** value in the range of 0-6. If there is no loss in the network, the quality of the decoded image is the same as that in the normal mode of operation. The quality of the decoded image changes only in the event of cell loss in the network.

We compare the quality of the resultant image when all the low priority cells are lost in the network due to congestion for various values of the **Quality** parameter. Assuming that only the low priority cells are completely lost in the network, and we can "reserve" bandwidth for all the high priority cells, the quality of the image will be least when **Quality** is 0 and maximum when the **Quality** value is 6. In this experiment, we do not use any spatial layering. The image is assumed to have no centre of interest.

Fig 24 shows the worst case scenario when all the low priority cells are lost due to network congestion for a **Quality** value of 0. In this case, the compressed image was spread over 157 cells and 80% of the compressed image constituted low priority cells. We see that the image reconstructed using only 20% high

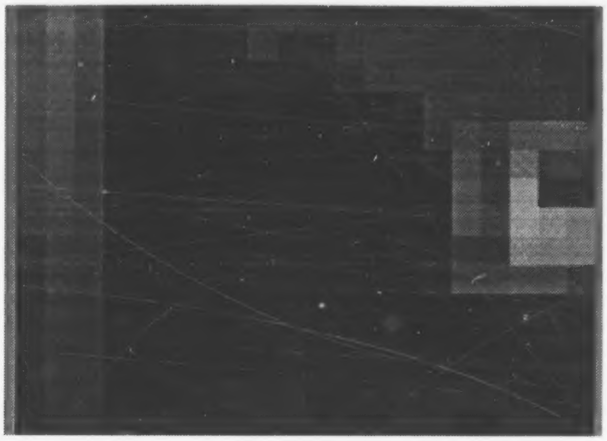


Figure 24: Image quality using frequency layering (Quality=0)

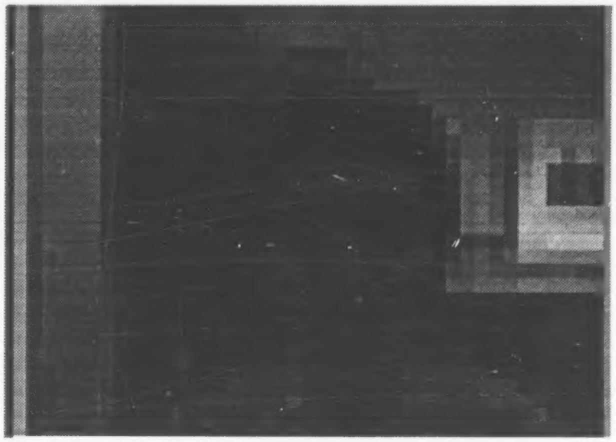


Figure 25: Image quality using frequency layering (Quality=1)

priority cells is *blocky* and lacks detail and sharpness due to the loss of high frequency components in the blocks.

Fig 25 shows the scenario when all the low priority cells are lost due to network congestion for a **Quality** value of 1. In this case, the compressed image was spread over 144 cells and 73% of the compressed image constituted low priority cells. We see that the image reconstructed using 27% high priority cells has better quality than the earlier case when we selected a **Quality** value of 0.

If the **Quality** value is 6, the decoded image will be of best quality in the event of loss of only low priority cells in the network. The decoded image quality is the same as in the case of *normal mode* of operation. This is due to the fact that all the block frequencies are sent as high priority data when the **Quality** value is set as 6.

The advantage of frequency layering is in limiting cell loss to high frequency coefficients of the video blocks. This makes the video codec more robust as compared to **Case 1**, where every portion of the image was equally susceptible to cell loss. An added advantage of layering is in sending vital information like the *end-of-frame-marker* as high priority data. This ensures that a frame is not lost due to loss of the marker alone, as could be the case in the *normal mode* of operation.

Image quality in spatial and frequency layered mode (Case 3): If Spatial Layering is also used, the quality of the image in the event of cell loss is better than in **Case 2**. It must again be noted that we consider network loss to be restricted to low priority cells. Here, we are able to localize cell loss to regions of the image which are of least subjective importance.

In this analysis, the fovea was selected at the centre of the image. 40% of the image around the fovea was considered to be subjectively important. We compare the quality of the decoded image when all the low priority cells are lost in the network due to congestion.

Even in the worst case (when **Quality** is set to 0), the decoded image is of better quality than in **Case 2**. For this case, the compressed image was spread over 152 cells. Also, 71% of the compressed image comprised low priority cells. The image reconstructed using 29% high priority cells is shown in Fig. 26. It is seen that the area around the fovea does not suffer any information loss. Loss is restricted to the high frequency components of the background information alone. Hence, only the background information appears *blocky*.

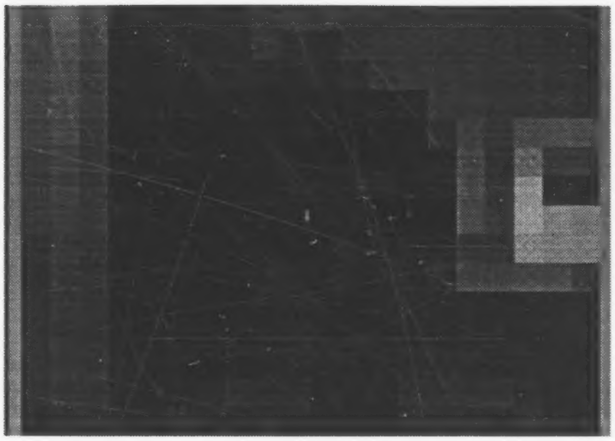


Figure 26: Image quality (Spatial and frequency layering)

If the **Quality** value is 6, the decoded image will be of best quality in the event of loss of all low priority cells in the network. All the block frequencies are sent as high priority data. Hence the decoded image quality approaches the quality in the *normal mode* of operation.

3.7.1 Bit-rate-penalty in SPAFLAY's *prioritized mode*

The layered coding algorithm used in SPAFLAY's *prioritized mode* increases the robustness of the video codec to cell loss. However, the advantage is at the expense of incurring a bit-rate-penalty compared to one-layer approaches like SPAFLAY's *normal mode*. This is due to an increase in the redundancy of the layered structure. In this section, we present a quantitative measure of the extent of bit-rate-penalty that we incur in the layered scheme used in SPAFLAY.

A typical *talking head* video scene is used in this analysis. The image is of size 120 x 160. This image is compressed using the various options in SPAFLAY. In each case, we compare the compressed image size with that in the *normal mode* of operation. This gives an estimate of the penalty incurred in the *prioritized mode* of operation. We also note the proportion of high priority data to the low priority data in the compressed video image in each case.

Case	Total bytes/frame	high priority bytes	low priority bytes
Normal mode	6936	6936	0

Table 3.2: Compressed frame size: *Normal mode*

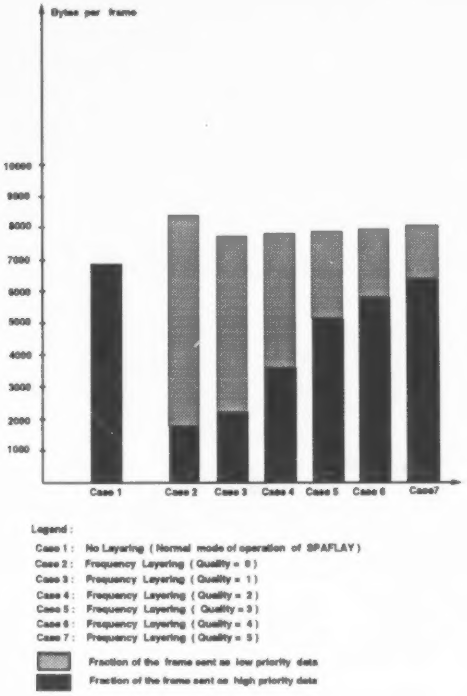


Figure 27: Bit-rate-penalty in frequency layering

Table 3.2 gives the size of the compressed frame in bytes when the *normal mode* of operation is used in SPAFLAY. This size (6936 bytes) serves as the reference while comparing the bit-rate-penalty incurred in the *prioritized mode.* We also note that in the *normal mode*, the entire frame is sent as high priority data.

Next, the initial image is compressed using frequency layering alone in SPAFLAY's *prioritized mode.* We note the size of the compressed image and the proportion of high and low priority data for various values of the **Quality** parameter. As the value of the **Quality** parameter increases, the proportion of high priority data in the compressed frame also increases. **Table 3.3** presents the values for these cases. It is illustrated in Fig. 27.

The point $(x = 80, y = 60)$ corresponding to the centre of the image, is selected as

Case	Total bytes/frame	high priority bytes	low priority bytes
Quality = 0	8461	1748	6713
Quality = 1	7747	2225	5522
Quality = 2	7851	3604	4247
Quality = 3	7951	5074	2877
Quality = 4	7999	5815	2184
Quality = 5	8049	6480	1569

Table 3.3: Compressed frame sizes: Frequency layering

Case	Total bytes/frame	high priority bytes	low priority bytes
Quality = 0	8416	1948	6468
Quality = 1	7729	2410	5319
Quality = 2	7830	3726	4104
Quality = 3	7921	5119	2802
Quality = 4	7968	5845	2123
Quality = 5	8015	6490	1525

Table 3.4: Compressed frame sizes: Spatial (20 %) + freq. layering

the *fovea*. We use spatial and frequency layering over the image. The area around the fovea which is subjectively important to the user is specified in terms of percentages- 20, 40 and 60. For each case, the **Quality** parameter values are again varied. These results are shown in **Table 3.4, 3.5, and 3.6**. They are also graphically presented in Figs. 28, 29 and 30 respectively.

From the results, it is clear that as the percentage of blocks subject to spatial layering increases, we incur less bit-rate-penalty. This is because the percentage of blocks subject to frequency splitting decreases. Hence, we approach the *normal mode* of operation values. However, this also increases the proportion of prioritized data in the compressed video frame. A substantial increase in the proportion of prioritized data jeopardizes the effectiveness of layering. If the percentage of blocks subject to spatial layering is decreased, the bit-rate-penalty increases. But this decreases the proportion of prioritized data in the compressed video frame. Thus, there is a tradeoff associated with decreasing the bit-rate-penalty and decreasing the proportion of prioritized data in a compressed frame.

Case	Total bytes/frame	high priority bytes	low priority bytes
Quality = 0	8242	2424	5818
Quality = 1	7632	2823	4809
Quality = 2	7717	3965	3752
Quality = 3	7796	5525	2571
Quality = 4	7838	5907	1931
Quality = 5	7881	6516	1365

Table 3.5: Compressed frame sizes: Spatial (40 %) + freq. layering

Case	Total bytes/frame	high priority bytes	low priority bytes
Quality = 0	7754	4344	3410
Quality = 1	7357	4582	2772
Quality = 2	7418	5255	2163
Quality = 3	7471	6006	1465
Quality = 4	7496	6386	1110
Quality = 5	7521	6720	801

Table 3.6: Compressed frame sizes: Spatial (60 %) + freq. layering

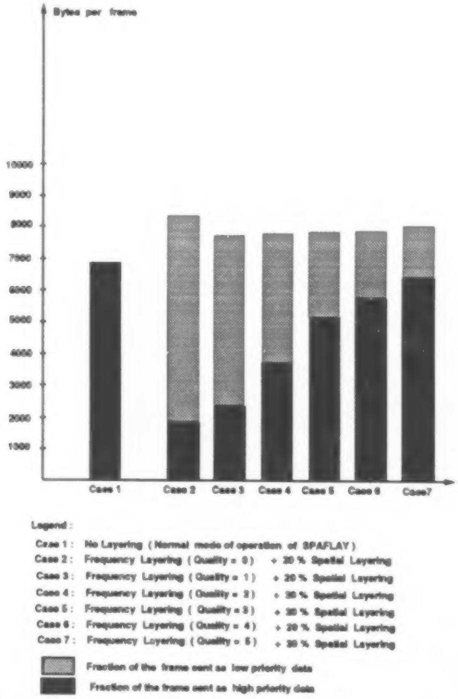


Figure 28: Bit-rate-penalty: Spatial (20 %) + freq. layering

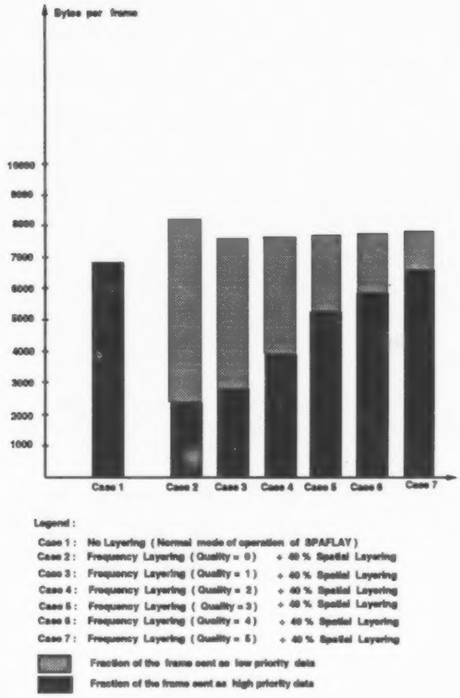


Figure 29: Bit-rate-penalty: Spatial (40 %) + freq. layering

3.7.2 Image quality in SPAFLAY

Another parameter necessary to evaluate the performance of SPAFLAY is the decompressed image quality. In particular, we are interested in the quality of the image when all the low priority cells are lost in the network.

Quality of an image is usually measured in terms of the signal-to-noise ratio of the image. However, a measure such as this, is inappropriate for an encoding scheme based on spatial importance like SPAFLAY. In these schemes, more importance is given to the area around the fovea than to the periphery. Hence, a measure such as the signal-to-noise ratio which treats the entire image uniformly is not suitable.

In SPAFLAY, we can quantify the quality of the decompressed image using a measure of the *number of motion blocks* transmitted for a video scene. At the encoding end, the high priority data component of a compressed frame is decompressed and used as a *reference frame* to detect areas of motion for the next captured frame. In

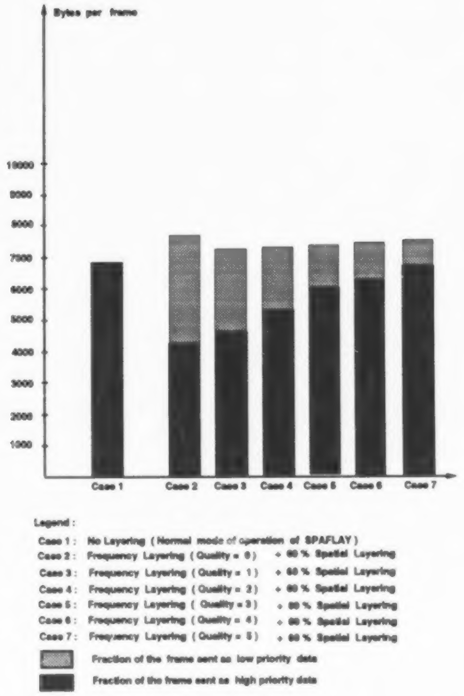


Figure 30: Bit-rate-penalty: Spatial (60 %) + freq. layering

the normal mode of operation of SPAFLAY, the entire compressed data of a video frame has uniform priority. Hence, when this frame is decompressed and used as a reference for the next frame, we detect only the areas of motion.

However, same is not the case in the prioritized mode of operation. Since only high priority data of a compressed frame is used as a reference, we detect not only blocks which constitute motion, but also blocks which are *coarse* and lower in quality to the corresponding block of the captured video frame. These coarse blocks are also spuriously accounted as *motion blocks.* If we increase the percentage of high priority data in the compressed video frame by either increasing the value of the **Quality** parameter or by specifying a greater area around the fovea as being subjectively important, then the number of *motion blocks* transmitted decreases. This is because the quality of the decompressed image increases. Hence, the blocks which actually constitute motion are alone detected as motion blocks and we decrease the number of spuriously detected motion blocks.

Case	No. of motion blocks	Avg. bpf	Peak bpf
Normal mode	22966	40400	51136
Q=0, S=0%	24454	53045	62968
Q=2, S=0%	23273	46908	57504
Q=3, S=0%	22189	45027	60016
Q=5, S=0%	22163	45495	60744

Table 3.7: Compressing a sequence of images

It should be noted that this measure of the number of motion blocks transmitted for a given video scene also gives an account of the redundancy incurred in the prioritized mode of operation. Only the high priority data is used as a reference to estimate motion. In effect, at the encoder's end, we assume the worst case, wherein all the low priority data is lost in the network. This pessimistic approach to motion estimation gives rise to redundancy in a layered encoding scheme due to the fact that blocks which do not actually constitute motion are also detected as *motion blocks*.

A typical teleconference video scene was filmed. This sequence of video frames was compressed using the various options in SPAFLAY. In each case, we note the number of motion blocks transmitted as also the peak and average bit-rate in terms of bits per frame (bpf). The results are presented in **Table 3.7**. We find that the number of motion blocks transmitted decreases as the value of the **Quality** (Q) parameter increases. This decrease is attributed to the increased quality of the decompressed image which serves as the reference frame for motion detection.

The proportion of high priority data in the compressed video stream measured over the entire filmed sequence for the various options in SPAFLAY is shown in Figs. 31 to 54. We select a particular image from the decompressed video stream to assess the quality of the image in each case when all the low priority cells are lost in the network. This image is presented alongside the graphs.

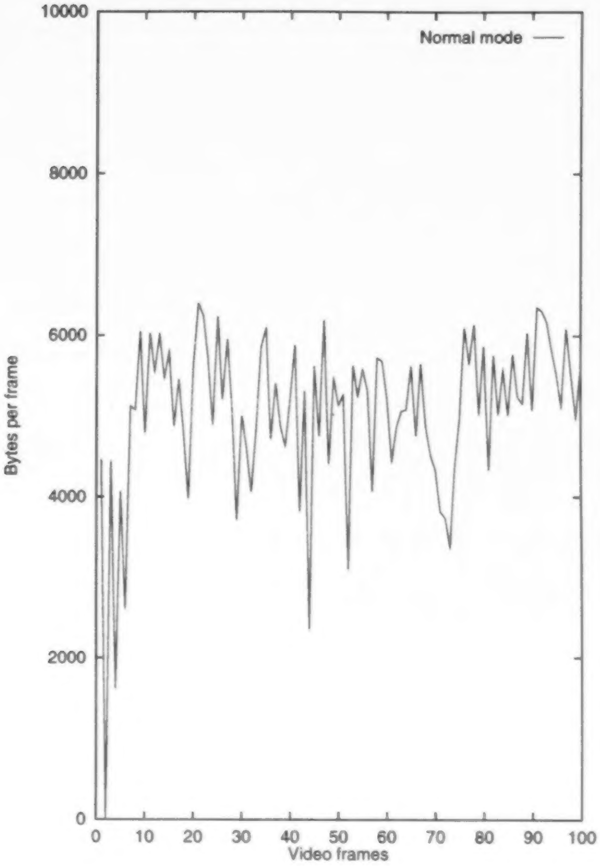


Figure 31: Case: *Normal mode*

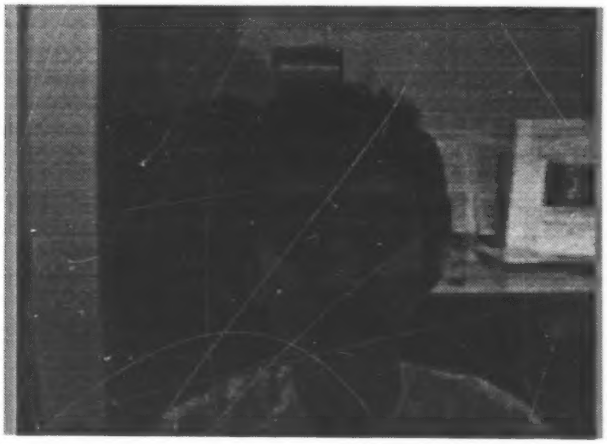


Figure 32: *Normal mode*: Decompressed image

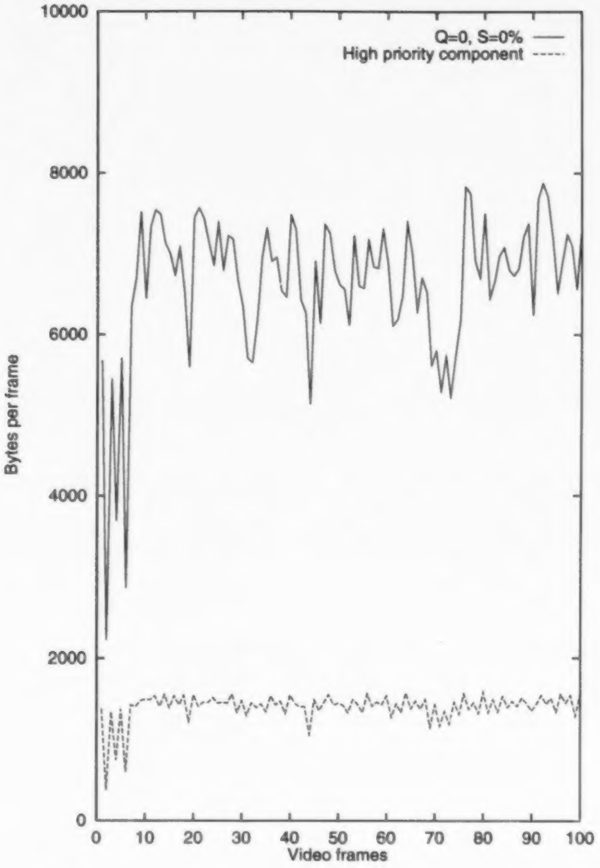


Figure 33: Case: Quality = 0 and Spatial(0%)

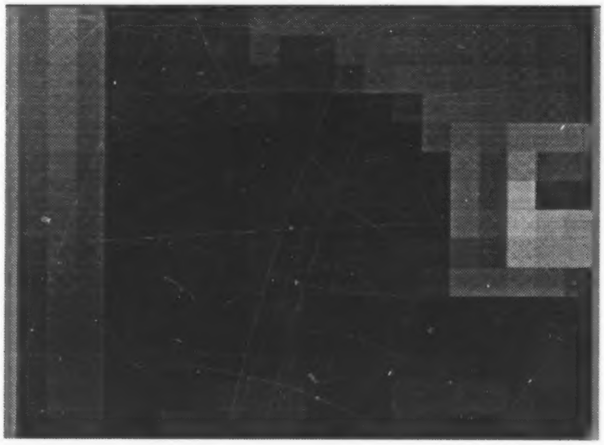


Figure 34: Quality = 0 and Spatial(0%): Decompressed image from HP cells

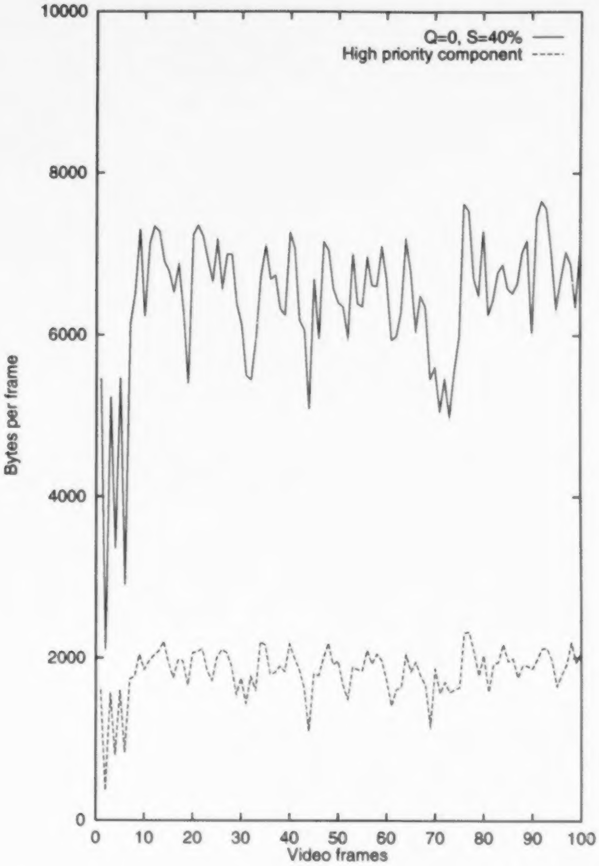


Figure 35: Case: Quality = 0 and Spatial(40%)

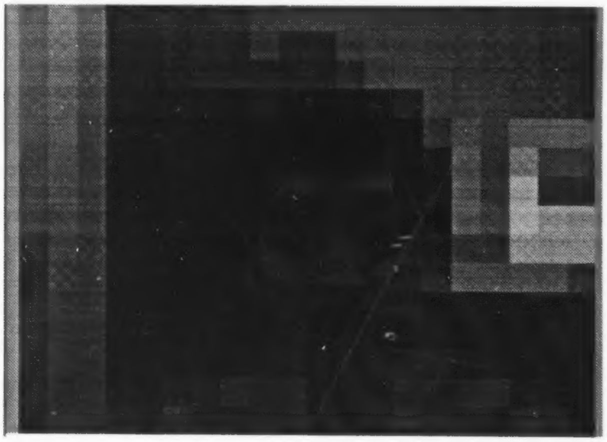


Figure 36: Quality = 0 and Spatial(40%): Decompressed image from HP cells

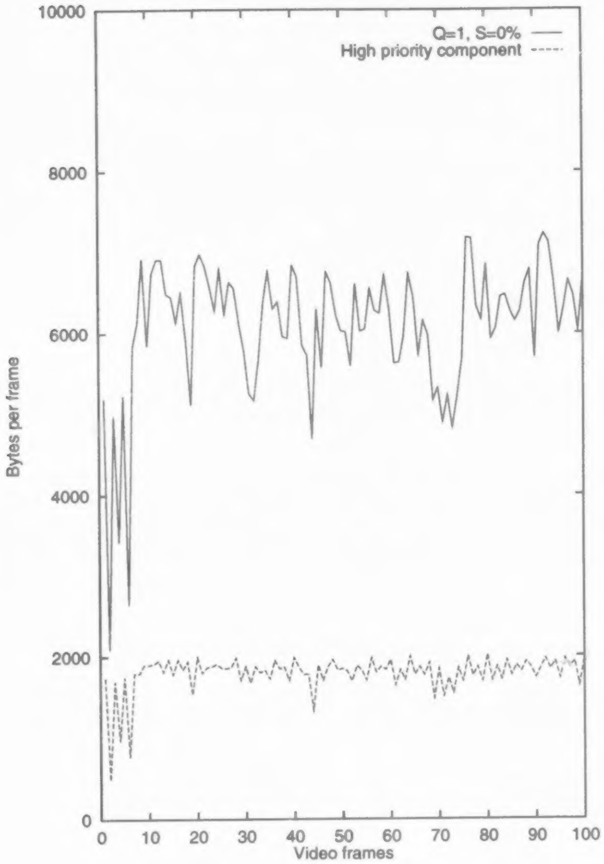


Figure 37: Case: Quality = 1 and Spatial(0%)

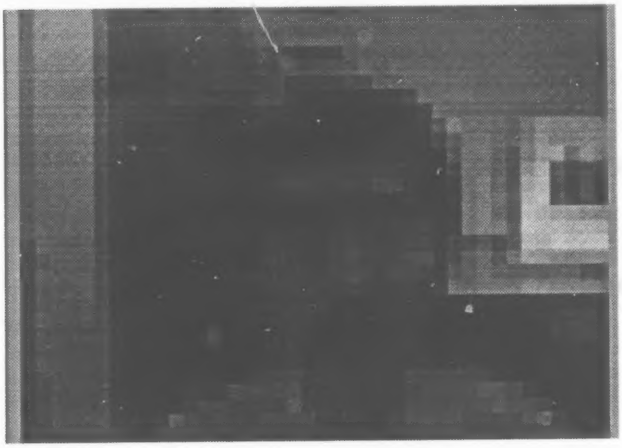


Figure 38: Quality = 1 and Spatial(0%): Decompressed image from HP cells

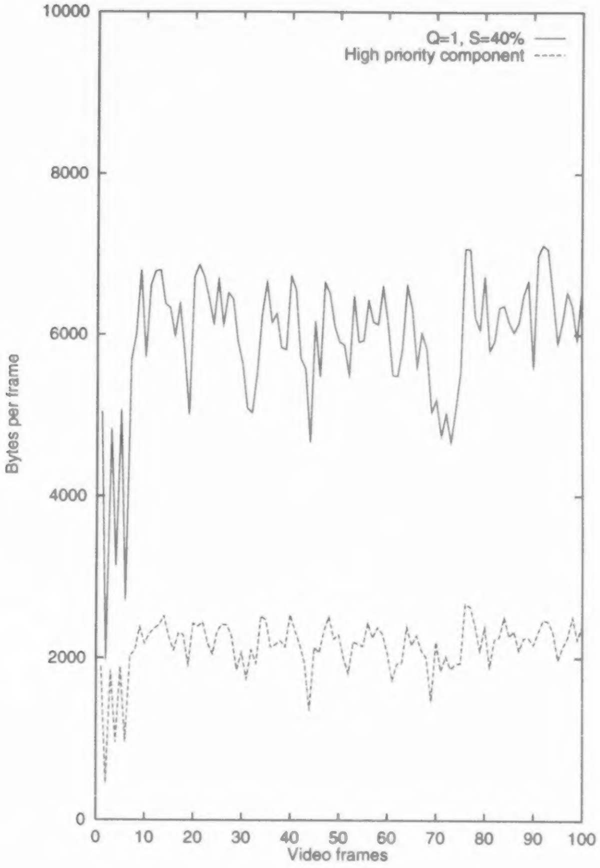


Figure 39: Case: Quality = 1 and Spatial(40%)

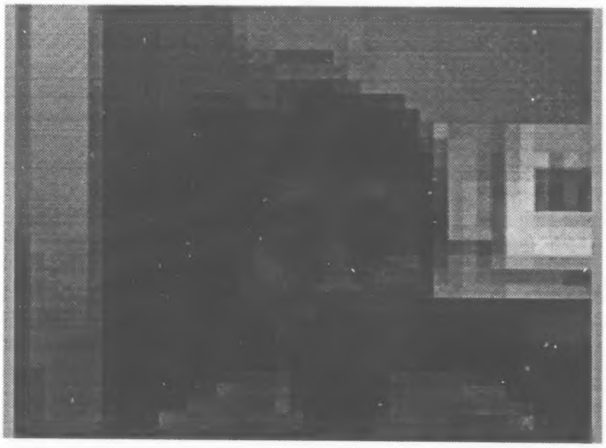


Figure 40: Quality = 1 and Spatial(40%): Decompressed image from HP cells

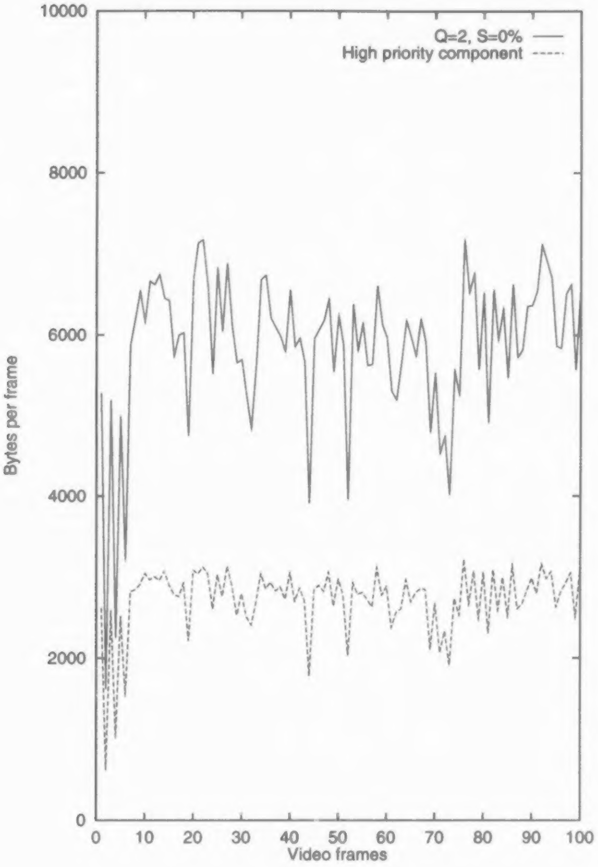


Figure 41: Case: Quality = 2 and Spatial(0%)

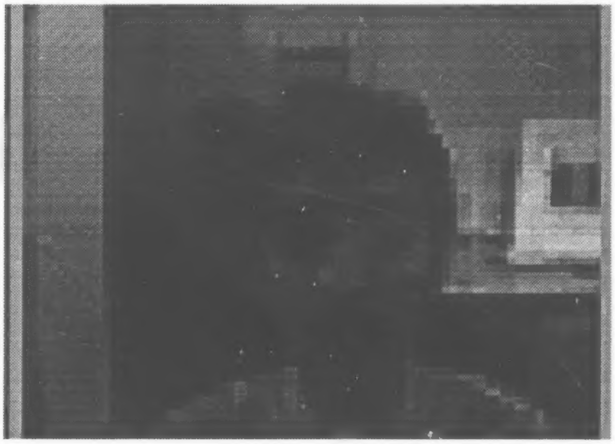


Figure 42: Quality = 2 and Spatial(0%): Decompressed image from HP cells

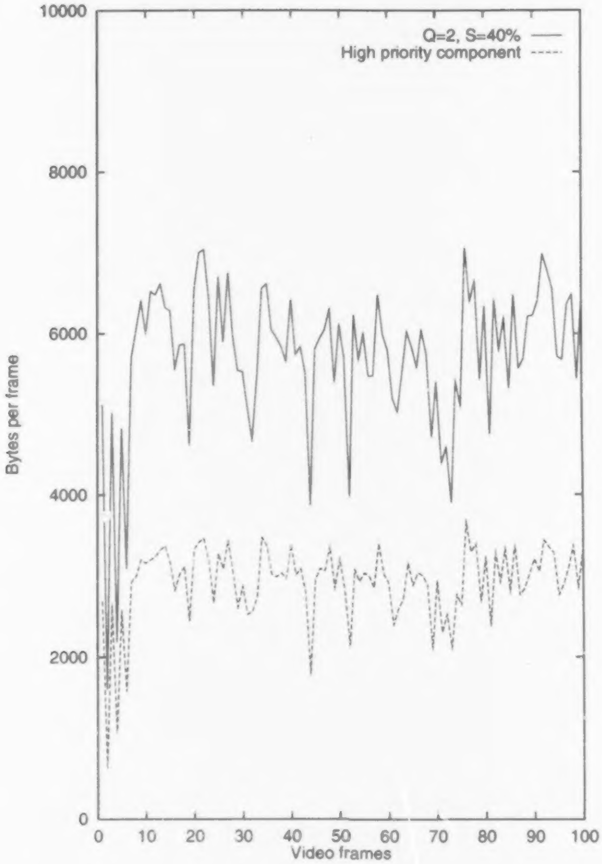


Figure 43: Case: Quality = 2 and Spatial(40%)

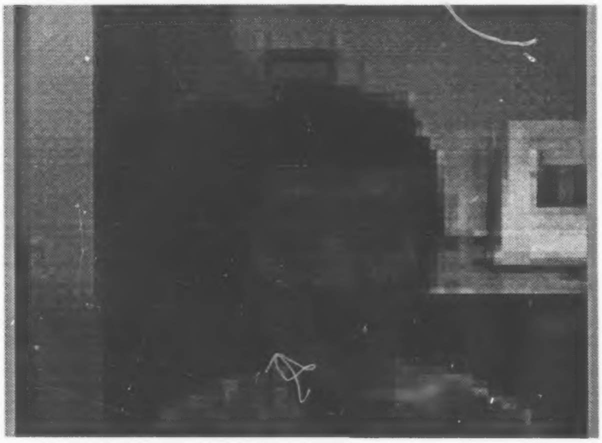


Figure 44: Quality = 2 and Spatial(40%): Decompressed image from HP cells

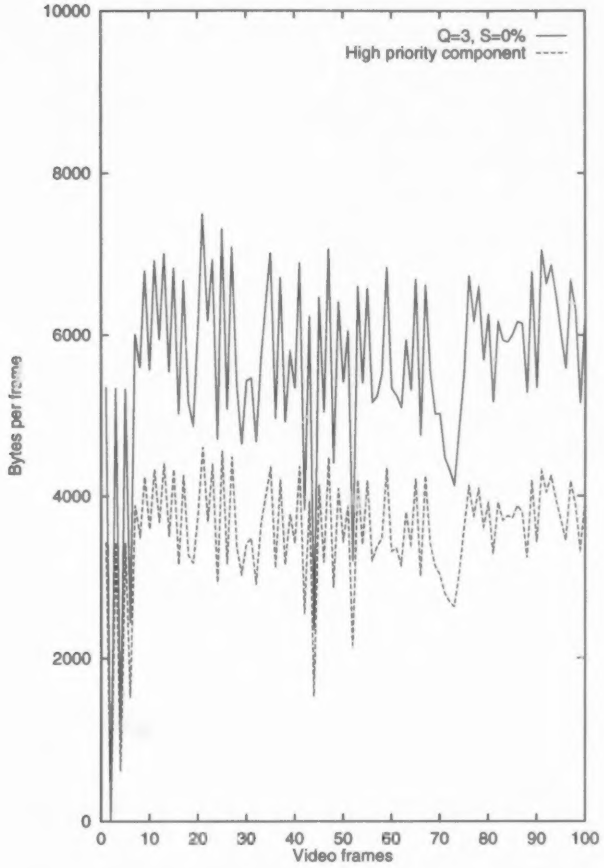


Figure 45: Case: Quality = 3 and Spatial(0%)

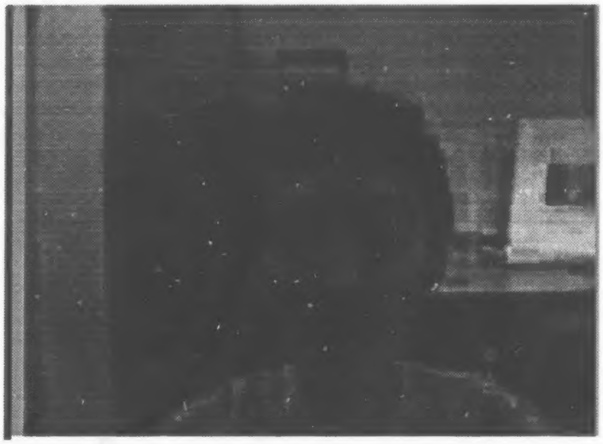


Figure 46: Quality = 3 and Spatial(0%): Decompressed image from HP cells

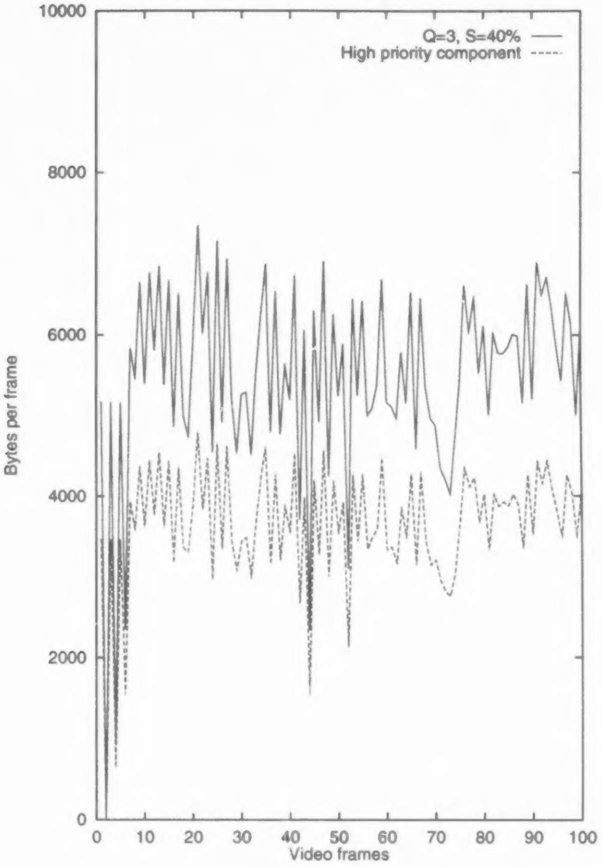


Figure 47: Case: Quality = 3 and Spatial(40%)

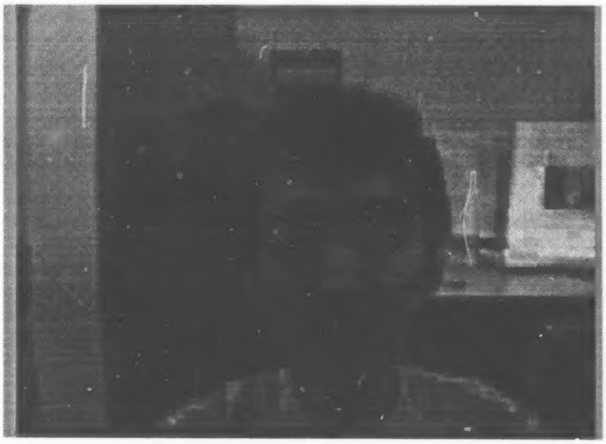


Figure 48: Quality = 3 and Spatial(40%): Decompressed image from HP cells

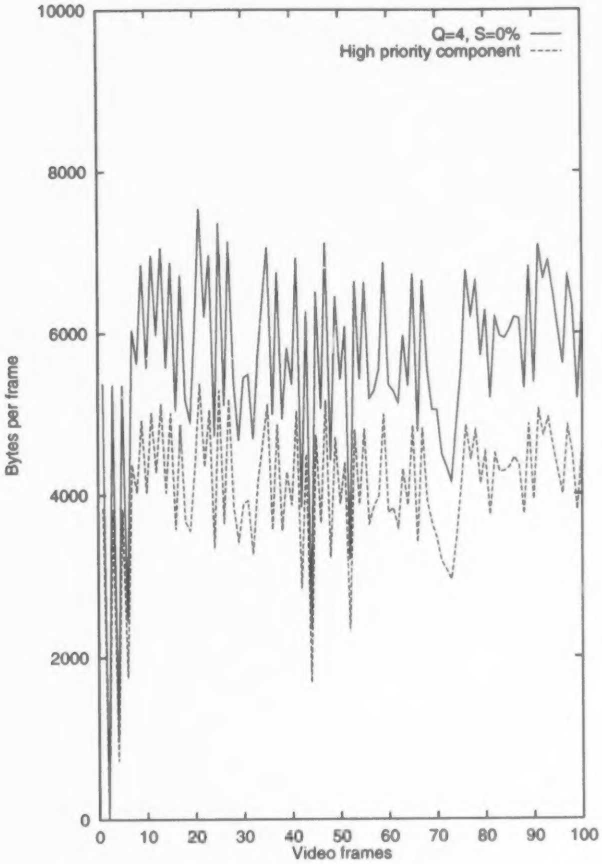


Figure 49: Case: Quality = 4 and Spatial(0%)

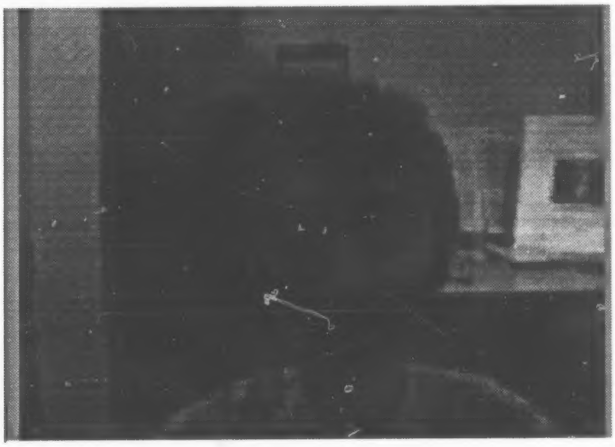


Figure 50: Quality = 4 and Spatial(0%): Decompressed image from HP cells

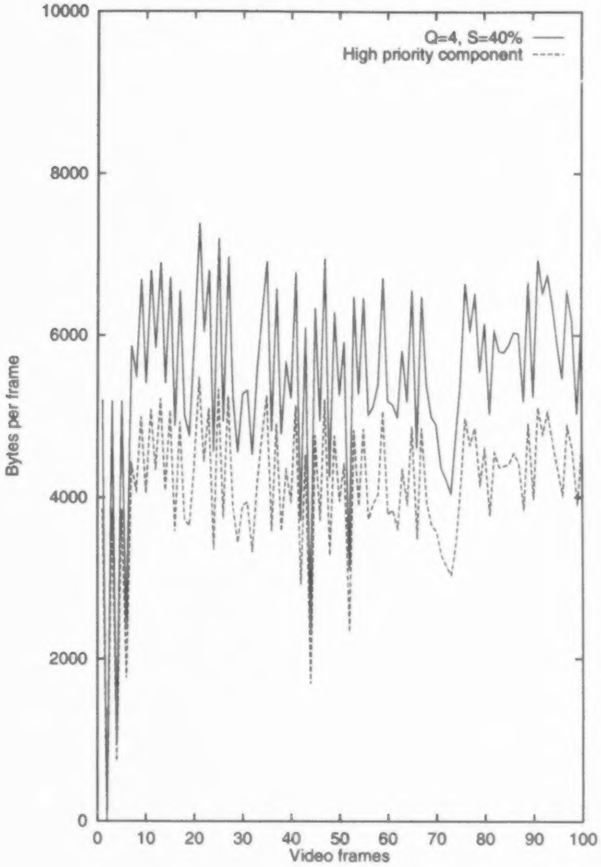


Figure 51: Case: Quality = 4 and Spatial(40%)

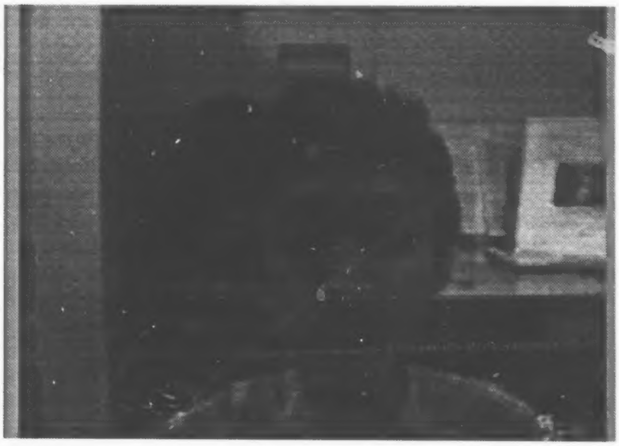


Figure 52: Quality = 4 and Spatial(40%): Decompressed image from HP cells

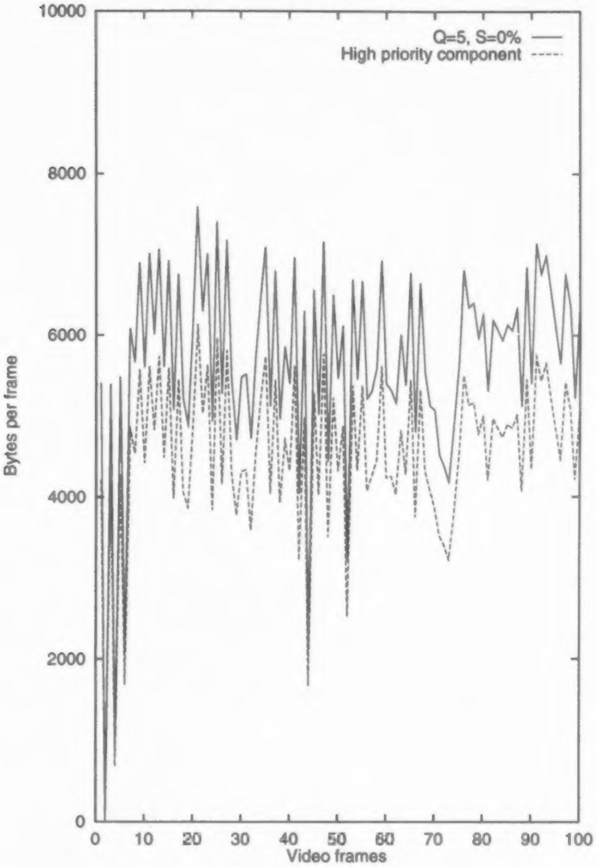


Figure 53: Case: Quality = 5 and Spatial(0%)

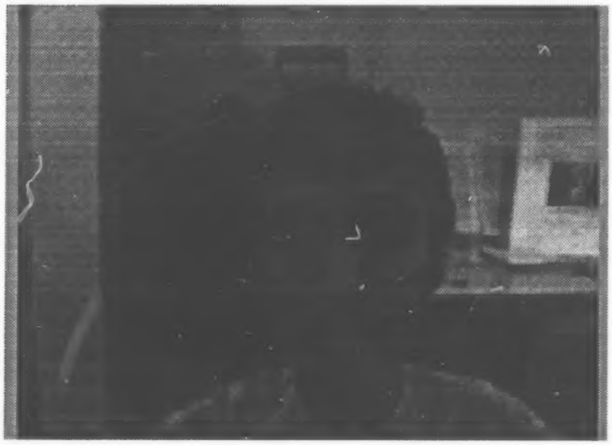


Figure 54: Quality = 5 and Spatial(0%): Decompressed image from HP cells

Chapter 4

Media synchronization

4.1 Temporal synchronization in multimedia applications

Synchronization of multiple data streams in time has been recognized as a significant requirement of future multimedia applications utilizing broadband communication technology. One of the requirements of any system supporting time dependent data is the need to provide synchronization of data elements which experience random delays during transmission and retrieval. In addition to the synchronization of *live* data streams that have an implied temporal relationship, we need synchronization for stored data elements of any media, including audio, video, text etc.

There are many issues involved in media synchronization for a networked multimedia application. For supporting the transmission of time dependent multimedia data over the network, real-time communication protocols are necessary. In our application, we have used Real-time Transport Protocol (RTP) [34] as the Transport Layer protocol. The merits of using RTP will be explained in a later section. Also, provision of a real-time operating system is essential to schedule time dependent tasks and assigning utmost priority to meeting deadlines. The real-time deadlines consist of playout times for the individual data elements. The design of a multimedia system of this nature must also account for latencies in each system component used in the delivery of data, from the source to the destination. We must use a smoothing mechanism to compensate for these latencies in order to provide synchronized playback of multimedia data.

Multimedia applications [1] are principally of three types:

Multimedia document browser: This lets the user select portions of a multimedia document, stored on a remote file system, for display. Each portion may consist of several parallel streams of audio and video. Graphic controls enable the user to pause, adjust audio volume, and so on.

Telephony and video conferencing system: This lets two or more users hold an audio or audio/video conversation with one another. This facility could also be integrated with group-oriented software, allowing participants to work together on a document.

Multitrack audio/video editor: It stores audio and video segments on one or more file servers. The segments can be played back, sequenced, and/or overlapped, according to an editable *play list*. New material can be recorded and overdubbed.

All these applications need a facility for I/O of continuous media data to and from a user's workstation. They also need synchronization of the different media streams so that data units which were captured at the same real-time are also displayed in unison.

A parameter which assumes significance in temporal synchronization is *skew*. Skew is defined as the difference in the presentation times of two related objects (i.e., video stream and audio stream). Coarse skew represents gross delays between an image and its accompanying voice, whereas fine skew represents the time delays between lip motion and voice. Skew objectives of various multimedia applications are given in **Table 4.8** [33].

Application	**Skew objective**
Audio + text or still image (one-way session)	Coarse skew < 1 sec
Audio + video (multipoint-to-multipoint sessions)	Coarse skew < 200 ms Fine skew: Audio in advance of video < 20 ms Video in advance of audio < 120 ms
Complex teleconferency	Coarse skew < 200 ms
Audio + video + still image + text	Fine skew: Audio in advance of video < 20 ms Video in advance of audio < 120 ms

Table 4.8: Skew objectives in multimedia applications

Skew arises due to the following reasons:

- The transmission start times and the network delays (Delay Jitter) may differ. This would skew the display of the data streams.

- Due to the pipelined decompression architecture, the video device might not start displaying until data for an entire frame is received (i.e., a frame represents a synchronization unit for video). On the other hand, an audio device might start playing out audio after a fixed number of audio samples representing a talkspurt is received. If an audio talkspurt is available at the destination, before an entire video frame can be reconstructed, the starting times for audio and video might be different and this could skew the output.

- The actual rates at which each physical device displays data are typically determined by separate quartz crystal oscillators. If the oscillator frequencies do not exactly match, audio and video output will not stay in synchronism over long periods.

- Scarcity of system resources (for example, network bandwidth and CPU time) may cause a stream to *starve*. If I/O continues on the other streams, the skew in the output will increase.

  While skew applies primarily to related media objects, samples of a single medium object are also affected by Delay Jitter. Delay Jitter causes these medium samples to be delayed by variable amounts over the network. Hence, playback of a single medium object must also account for Delay Jitter.

4.2 Accounting for Delay Jitter

Though there are numerous reasons for skewed output between data streams, delay variance in the network or jitter is the single most prominent reason in distributed multimedia applications.

In ATM networks, the end-to-end delay of the ith cell is given as $D + W_i$, where D is a constant that includes the propagation, transmission plus the switching delays and W_i is the random delay component that arises out of buffering within the network. The interarrival time of cells at the receiver is given by (4.1).

$$(D + W_{i+1}) - (D + W_i) = \delta \tag{4.1}$$

Ideally, interarrival times of cells at the receiver is equal to the time interval by which the cells were separated at the sender. This is the case if $W_{i+1} = W_i$. However, due to the randomness in the network, W_i is a random variable and is not a constant. δ represents the Delay Jitter.

Hence, for temporal synchronization of data streams, we need some form of *smoothing* mechanism at the destination to account for Delay Jitter [23]. The goal of this technique is to reshape the distribution of arriving cells of a data stream to reduce delay variance. This process is shown schematically in Fig. 55, where $p(t)$ is the delay density function and $w(t)$ is the reconstructed playout distribution.

Jitter can be controlled at the receiver at the expense of large buffers and delaying cells. We could temporarily store the arriving cells in a jitter removal buffer so that the departure rates of the cells from the buffer are close to the interexit times of cells at the sender as illustrated in Fig. 56. In our application, we use this technique of buffering media data before playback to smoothen out the effect of Delay Jitter [22]. This will be explained in detail in the section on **Intra-medium synchronization**.

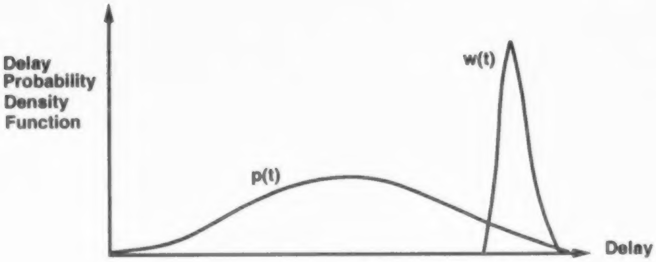


Figure 55: Reduction of arriving cell delay variance

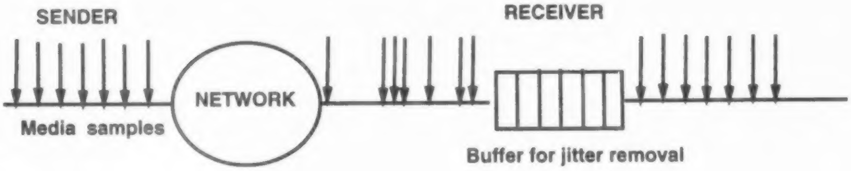


Figure 56: Buffering to handle jitter

4.3 Synchronization options

Synchronization can be divided into two categories: **intra-medium** synchronization and **inter-media** synchronization. Continuous media data streams are defined as a sequence of *units* (video frames or audio samples). Intra-medium synchronization [35] refers to playing out these data units within a medium stream at appropriate times. Inter-media synchronization [1] is related to correlating data units belonging to different media streams and playing out these data units in unison.

Our temporal synchronization model allows clients to specify how to synchronize output. We provide two models for intra-medium synchronization: *Blind-Timing* and *Absolute-Timing*. The client has to specify one of these two techniques at the start of a multimedia session. Both Blind-Timing and Absolute-Timing models for intra-medium synchronization have their merits and drawbacks. In a later section, we will examine the suitability of selection of one of these models depending on the nature of the application at hand. Inter-media synchronization in our application is based on a *master-slave* synchronization model. In this model, one of the media streams is designated as the *master* and the data units of other media streams are correlated

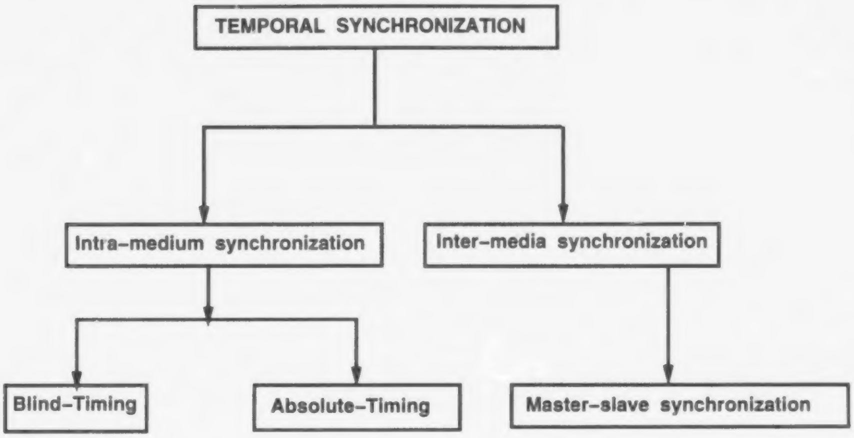

TEMPORAL SYNCHRONIZATION
Intra–medium synchronization
Inter–media synchronization
Blind–Timing
Absolute–Timing
Master–slave synchronization

Figure 57: Classification of temporal synchronization techniques

and synchronized with the master stream's data units. This synchronization model is used for playback of orchestrated applications and for live applications. The temporal synchronization techniques in our multimedia application are classified in Fig 57.

4.4 Intra-medium synchronization

In multimedia playback applications, the source takes some signal, packetizes it, and then transmits the packets over the network. Jitter is introduced by the network due to variable queuing delays. The receiver depacketizes the data and then attempts to faithfully playback the signal. This is done by buffering the incoming data to remove the network induced jitter and replaying the signal at some fixed offset delay from the original departure time. The term *playback point* refers to the point in time which is offset from the original departure time by this fixed delay. Any data that arrives before its associated playback point can be used to reconstruct the signal. Data which arrives after the playback point is essentially useless in reconstructing the real-time signal.

In order to choose a reasonable value for the offset delay, an application needs some *a priori* characterization of the maximum delay its packets will experience. This *a priori* characterization could either be provided by the network in a quantitative service commitment to a delay bound, or through the observation of the delays experienced by the previously arrived packets. The application needs to know what delays to expect, but this expectation need not be constant for the entire duration of the flow.

The performance of a playback application is measured along two dimensions: *latency* and *fidelity* [36]. In general, latency is the delay between the two ends of a distributed application; for playback applications, latency is the delay between the time the signal is generated at the source and the time the signal is played back at the receiver. Applications greatly vary in their sensitivity to latency. Some playback applications, in particular, those that involve interaction between the two ends of a connection are more sensitive to latency. Playback applications like transmitting a movie or lecture are not sensitive to this parameter to the same extent.

Fidelity is the measure of how faithful the playback signal is to the original signal. The playback signal is incomplete when packets arrive after the playback point and are dropped rather than played back. The playback signal becomes distorted when the offset delay is missed. Therefore, there is a decrease in fidelity when packets miss their playback point. Also, if the offset delay is varied, there is again a decrease in fidelity.

Delay can affect the performance of playback applications in two ways. First, the value of the offset delay, which depends on predicting the future packet delays, determines the latency of the application. Second, the delays of individual packets can decrease the fidelity of the playback by exceeding the offset delay; the application then can either change the offset delay in order to playback packets late (which introduces distortion) or merely discard the late packets (which creates an incomplete signal). These two ways of coping with late packets offer a choice between an incomplete signal and a distorted one, and the optimal choice often depends on the details of the application. But we must note that late packets necessarily decrease fidelity.

Applications which are intolerant to even some loss of fidelity (*intolerant applications*) must use a fixed offset delay, since any variation in the offset delay will introduce some distortion in the playback. For a given distribution of packet delays, this fixed offset delay must be larger than the absolute maximum delay in the network, to avoid the possibility of late packets. On the other hand, applications which are less sensitive to a little loss of fidelity (*tolerant applications*) need not set their offset delay greater than the absolute maximum delay, since they can tolerate some late packets. These applications can also vary the offset delay to some extent while a session is in progress, as long as it doesn't create too much distortion.

Thus, tolerant applications have a much greater degree of flexibility in how they set and adjust their offset delay. These applications need not use a single fixed delay for the offset delay. They can try to reduce their latency by varying their offset delays in response to actual packet delays experienced in the recent past. These applications are called *delay-adaptive* playback applications. This adaptation is optimistic since it relies on past packet delays to account for future packet delays. When the application loses the gamble, there is a momentary loss of data as packets miss their playback points, but since the application is tolerant of such losses, the decreased offset delay may be advantageous. However, there is a complicated tradeoff between the advantage of decreased offset delay and the disadvantage of reduced fidelity due to the variations of the offset. In our application, we have used a fixed maximum offset delay. It decreases the chances of a medium packet missing its playback point and thus offers greater fidelity. But this technique increases the latency as also the buffering at the application.

Increase in the latency poses no serious drawback for the playback of orchestrated applications. There is also reason to believe that most playback applications will have sufficient buffering to store packets until their playback point. This is based on the fact that the storage needed is a function of the queuing delays, not the total end-to-end delay. Queuing delays for playback applications will not increase as the networks get faster. Also, since memory is getting cheaper, providing sufficient buffering before playback at the receiver's end will become increasingly practical.

4.4.1 Blind-Timing and Absolute-Timing

Before discussing the details of Blind-Timing and Absolute-Timing, we need to define the following terms:

Playout unit: A playout unit is a group of packets sharing a common timestamp. For voice, the playout unit would typically be a single voice segment (comprising of voice packets which were all grabbed at the same time instant), while for video, a frame could be broken into subframes, each consisting of packets sharing the same timestamp and ordered by some form of sequence number.

Synchronization unit: A synchronization unit consists of one or more playout units that, as a group, which share a common delay between generation and playout of each part of the group. The delay may change at the beginning of such a synchronization unit. The common synchronization units are talkspurts for voice and frames for video transmission.

Absolute and Relative timing: Two concepts related to synchronization and playout units are *absolute* and *relative* timing. Absolute timing maintains a fixed timing relationship between the sender and the receiver, while relative timing ensures that the local spacing between the packets at the sender and the receiver is the same.

Timestamp: Most proposed synchronization methods require a timestamp. The timestamp has to have a sufficient range so that wrap arounds are infrequent. A 32-bit timestamp is expected to serve all anticipated needs, even if the timestamp is expressed in units of samples or other sub-packet entities. A timestamp may be useful not only at the Transport, but also at the Network Layer, to schedule packets based on urgency.

The following variables also have to be defined to describe our synchronization model. A subscript n represents the nth packet in a synchronization unit, $n = 1, 2, \ldots.$ Let a_n, d_n, p_n and t_n be the arrival time, variable delay, playout time and generation

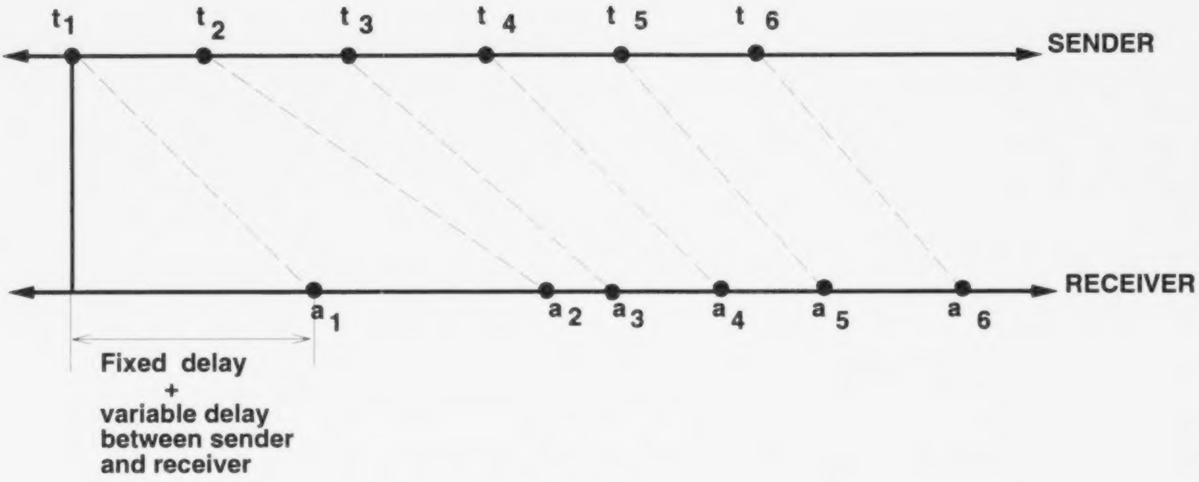


Figure 58: Playout synchronization variables

time of the nth packet, respectively. Let τ denote the fixed delay from the sender to the receiver. Let d_{max} denote the *estimated* maximum variable delay (jitter) within the network. The estimate is typically chosen in such a way that only a very small fraction (on the order of 1%) of the packets take more than $\tau + d_{max}$ time units. For best performance under changing network conditions, the estimate should be refined based on the actual delays experienced. The variable delays in a network consist of queuing and medium access delays, while the propagation and processing delays make up the fixed delay. An additional end-to-end fixed delay is introduced by packetization; the non real-time nature of most operating systems adds a variable delay both at the transmitting and the receiving end. The sender and the receiver clocks are assumed to run at the same speed for simplicity. The relationship between these variables is depicted in Fig. 58.

From the above definitions, we get the relationship:

$$a_n = t_n + d_n + \tau \tag{4.2}$$

This equation holds for every packet. We also define l_n as the *laxity* of packet n, i.e., the time $(p_n - a_n)$ between arrival and playout. Synchronization techniques differ only in how much they delay playback of the first synchronization unit of a medium stream.

We have incorporated two methods of intra-medium synchronization in our application: Blind-Timing and Absolute-Timing. In each technique, we compute the playback time for the first synchronization unit of a medium stream on the basis of the delay estimate d_{max}.

Blind-Timing : In this method, we assume that the first packet in a talkspurt experiences only the fixed delay, so that the full d_{max} value has to be added to allow other packets within the talkspurt experiencing more delay.

$$p_1 = a_1 + d_{max} \tag{4.3}$$

Blind-Timing does not require timestamps to determine p_1. It only needs an indication of the receipt of the first packet in a playback session. Subsequent packets are played out on the basis of the generation time interval of the packets. Theoretically, $c = (t_n - t_{n-1})$ should be a known constant. It represents the generation time interval between two samples of a medium. This is the case for a medium like audio, where the generation time interval between successive samples is a constant and known in advance. Hence, the medium samples are played out at times given by 4.4.

$$p_n = p_{n-1} + c \tag{4.4}$$

The playout times are also depicted in Fig. 59.

Absolute-Timing: Absolute-Timing is the playback technique that we should use when the generation time interval between samples of a medium is not a constant. In VBR video encoding, this is normally the case. A sudden increase in the motion content of a scene results in more video frame blocks having to be compressed and transmitted. In this case, the time interval between successive samples generated may vary.

Therefore, in Absolute-Timing, we calculate the exact generation time interval between successive packets of a medium. This interval is used to schedule the

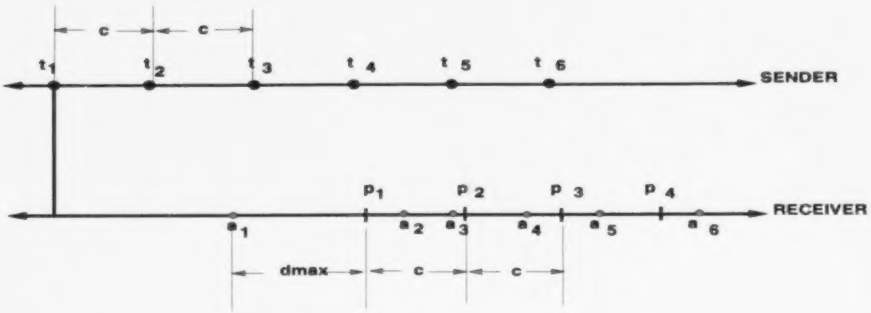

c
c
t_1
t_2
t_3
t_4
t_5
t_6
SENDER
p_1
p_2
p_3
p_4
RECEIVER
a_1
a_2
a_3
a_4
a_5
a_6
dmax
c
c

Figure 59: Blind-Timing intra-medium synchronization

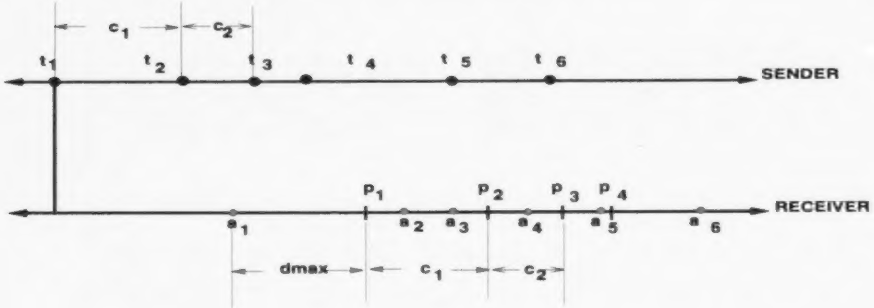

c_1
c_2
t_1
t_2
t_3
t_4
t_5
t_6
SENDER
p_1
p_2
p_3
p_4
RECEIVER
a_1
a_2
a_3
a_4
a_5
a_6
dmax
c_1
c_2

Figure 60: Absolute-Timing intra-medium synchronization

playout times of all samples. The first packet is played out as per Eq. 4.4 in the Blind-Timing case. Subsequent packets are played out based on the position of the first packet as follows:

$$p_n = p_{n-1} + (t_n - t_{n-1}) \quad for \quad n > 1 \tag{4.5}$$

The playout times are also depicted in Fig. 60. Both these techniques are available in our the application. The user is given a choice at the beginning of a session to select one of these playout techniques.

The d_{max} delay is also input as a parameter at the start of the video session. It signifies the amount of time that must elapse after the receipt of the first medium sample before the sample is played out.

Due to the non real-time nature of the operating system, in reality, the medium packets cannot be played out at exact intervals as desired. Absolute-Timing, by the very nature of the technique, imposes more demands on the operating system than Blind-Timing. In our implementation, it was observed that Absolute-Timing results in playback points being more widely separated from each other than required, because of the non real-time nature of the operating system. The conservative estimate used in Blind-Timing offers a better performance.

4.5 Inter-media synchronization

Inter-media synchronization is the temporal coordination of multiple streams (e.g., voice, video, graphics and text) relative to one another, and requires concurrent streams to be played out at identical synchronization times. During delivery of such data at the user's end, maintaining the required temporal association between points in the data across various streams is necessary in the presence of transport delays in the network. Data streams are split into application perceivable distinct units of data along the real-time axis and *temporal presentation control* is exercised on these data units at user's end. The temporal presentation involves extracting the timing relationship between the various data units collected at the source and determining the real-time interval and order in which these data units may be delivered to the destination for processing.

Inter-media synchronization can be done by the network or by the application. The idea of the network synchronizing the various data streams is not advisable since it reduces data transport flexibility and scalability. We would need to maintain extensive state information in the network about the various streams, particularly when the streams flow through different parts of the network. Hence, it is desirable to transmit streams independently even if the playout times of data units in the two streams may be identical. In our inter-media synchronization model, the onus is with the application in enforcing synchronization using the services of the lower layer.

Data streams possessing the same schedule in terms of playout times may not necessarily require identical offset delays. The streams can represent different classes of data in the network, and can have different fixed delay components due to differing data size. To maintain inter-media synchronization, each stream must experience the same overall delay to avoid skewing. For inter-media synchronization, the variable delays of each stream must be accommodated, i.e., each stream must be delayed as long as the largest anticipated variable delay.

The synchronization problem has two parts: *framing of data streams* which refers to identifying the various points in the data streams which need to be synchronized and *temporal presentation control* [31] which refers to ordering of various points in the data streams over real-time for delivery at users, as required in the application. In our solution approach, the temporal axis of an application is segmented into distinct intervals corresponding to the playout times of samples of one medium designated as the *master*. The playout times of the master stream are decided in accordance with one of the intra-medium synchronization techniques. The other media are *slaves* and are *dependent* on the *master* for their playback.

Framing of data streams and temporal presentation control are based on a **master-slave** relationship between the media streams. One of the media streams is designated as the master. The choice of the master stream [1] is based on the following considerations:

- The medium with the smallest interval between successive playout units is designated as the master.
- Some applications are more intolerant to loss in fidelity of a particular medium than others. This medium is chosen as the master. The overall effectiveness of the application depends on the quality of this medium. In an application involving video and audio, an occasional lapse in the quality of video is permissible. The application is more sensitive to lapses in the quality of audio. Hence, it is appropriate to select the audio stream as the master.

4.5.1 Framing of data streams

In our application, we need a synchronized playback of full motion video and audio. Points of synchronization between audio and video streams are identified at capture time. This framing information is stored in the headers of the video and audio data units. At the receiver's end, the framing information is extracted and is used to playout audio and video data in synchronism.

Framing information in video: Video is compressed using the SPAFLAY codec discussed in **Chapter 3**. In SPAFLAY, a video frame is initially subjected to **Block analysis** to detect the *motion blocks*. These blocks alone go through the complete compression procedure and are transported to the receiver's end. To increase the throughput of the video channel, more than one *motion block* is compressed and sent in a single *video data unit*. In our experiments, it was found that video data units of size 1 Kbytes was a good choice. Depending on the compression achieved on each video block, a video data unit normally contains about 4-5 compressed motion video blocks. As we increase the size of the video data unit, the throughput of the video channel increases, but the end-to-end delay also increases. Hence, there is a tradeoff between increasing the throughput of the video channel and decreasing the end-to-end delay.

Each video data unit has a header of the following structure:

```
struct rtphdr
{
  uint8   rh_chanid:6;    /* channel id */
  uint8   rh_vers:2;      /* version */
  uint8   rh_content:6;   /* content id */
  uint8   rh_sync:1;      /* end of synchronization unit */
  uint8   rh_opts:1;      /* options present */
  uint16  rh_seq;         /* sequence number */
  uint32  rh_ts;          /* time stamp (middle of NTP timestamp) */
}
```

Presently, only the fields **rh_seq** and **rh_sync** are of relevance in framing of data streams. The other fields will be explained in a later section on RTP. The flag **rh_sync** is set in the header of the last video data unit which is sent for a video frame. It is used as an *end-of-frame* marker. In all other video data units, this flag is unset. The field **rh_seq** in the header of a video data unit contains the sequence number of the video frame. All the video data units for a video frame have the same sequence number. The sequence number indicates the position of the current video frame with respect to others in this session. This field is also used in framing of video and audio streams.

At the decoding end, these video data units are received. The compressed motion blocks from the video data units are extracted and decoded. The entire video frame is reconstructed using these decoded blocks. If the flag **rh_sync** in the received video data unit is set, it is an indication to the decoder that the reconstructed frame is ready for display. Thus, the unit of synchronization for video is a frame. Typically, video frames have a maximum display rate of 30 frames/s.

Framing information in audio: If audio is sampled at 8Khz, it accounts for 8000 samples/s. Audio samples are also played out at this rate. Since the number of audio samples to be played out is greater than the number of video frames to be displayed in any given time interval, the audio stream is designated as the master. A captured video frame is associated/bound with the nearest audio sample in time. In this case, the synchronization between audio and video data streams will be fine.

To increase the network throughput for the audio channel, it is reasonable to group a set of audio samples as one data unit. Now, the video frame will be associated with an audio data unit rather than a single audio sample. It should be noted that as the size of the audio data unit increases, the synchronization between audio and video streams will become more coarse.

Hence, there is a tradeoff between increasing the throughput of the audio channel and decreasing the coarseness in synchronization between audio and video data streams. In our experiments, it was found that audio data units of size 300 bytes was a reasonable choice. The audio data unit contains 300 audio samples. Each audio data unit has a header structure as follows:

```
struct Audio_rtphdr
{
   short video_frame; /* whether there is a video frame associated with
                         this audio data unit */
   uint16 rh_seq; /* sequence number of the audio data unit */
   float rh_ts;   /* audio time stamp */
 }
```

The field **rh_seq** indicates the position of the current audio data unit with respect to other audio data units in the session. The flag **video_frame** is used to associate slave medium samples (video frames) with the master stream's samples (audio data units). If the flag is set, it indicates the presence of a video frame to be played out simultaneously with this audio data unit. The flag is cleared in all audio data units which do not have a video frame associated with them. The flag is of type **short**. In fact, a single bit is sufficient to indicate the presence of a video frame which is to be associated with the audio data unit. By having an entire field dedicated to this purpose, we allow the possibility of more than one medium being associated with the audio data unit. We can visualize other media like text, slow motion video etc. being also a part of this application. In this event, it will be very simple to associate the synchronization points of other media with an audio data unit.

The points of synchronization in our application are identified by *binding* a video frame to an audio data unit. This is done at capture time. By binding, we mean associating a common sequence number for a video frame and an audio data unit. Also, the audio data unit has a flag which is set to indicate the presence of a video frame which has to be played out in unison with this audio data unit at the receiver's end.

Binding audio and video framing information: Our application is based on a *Client-Server* model. The server also called the *sender*, captures audio and video data. It compresses and transmits this multimedia data over the network to connected clients. The clients decompress this data and playback the audio and video material. The server is associated with the task of framing of data streams while the client uses temporal presentation control to playback the audio and video streams in synchronism.

There are two modules at the sender/server which are associated with the task of capturing, compressing and transmitting data: the **Server-Audio-Module** and the **Server-Video-Module**. The framing of data streams is also managed entirely by these two modules. For the purpose of framing of data streams, the audio and video modules share a 16-bit unsigned integer variable named *current-audio-sequence-number*. The video module gets access to the sequence number of the last captured audio data unit through the value of *current-audio-sequence-number.*

At capture time, the **Server-Audio-Module** grabs audio samples of number equivalent to that in an audio data unit. A sequence number is associated with the audio data unit by incrementing the value of *current-audio-sequence-number.* This value is placed in the header of the audio data unit in the **rh_seq** field. This header is added to the audio data and sent over the network.

The video module grabs video frames and compresses it using the SPAFLAY codec. When a video frame is captured, it has to be *bound* to the nearest audio data unit. We associate the video frame with the next audio data unit to be captured. This is done by setting the **rh_seq** field in a video data unit header to the value *current-audio-sequence-number + 1.* With this, the video frame has the same sequence number as the next audio data unit to be grabbed. This value is replicated in all the video data units for the grabbed frame. The video module also signals to the audio module that a video frame has been captured.

When the next audio data unit is grabbed, all that needs to be done is to set the flag **video_frame** in the header. It indicates the presence of a video frame which shares the same sequence number as this audio data unit.

The functionality of the audio module is summed up in the block diagram shown in Fig. 61. The Sender-video-process functionality is presented in Fig. 62. The framing of data streams is carried out by associating a common sequence number for an audio data unit and a video frame. The audio data units which are bound to video frames also have the flag **video_frame** set. If there are no video frames associated with the audio data unit, then this flag is cleared. Binding of audio data units with video frames is shown in Fig. 63.

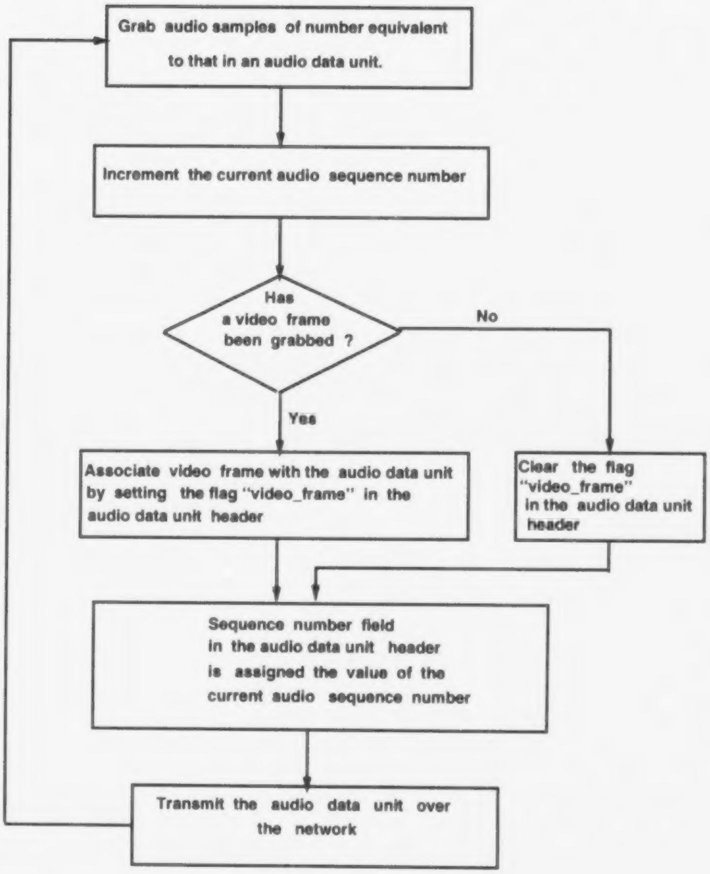

Grab audio samples of number equivalent to that in an audio data unit.
Increment the current audio sequence number
Has a video frame been grabbed ?
No
Yes
Associate video frame with the audio data unit by setting the flag "video_frame" in the audio data unit header
Clear the flag "video_frame" in the audio data unit header
Sequence number field in the audio data unit header is assigned the value of the current audio sequence number
Transmit the audio data unit over the network

Figure 61: Functionality of Sender-audio-process

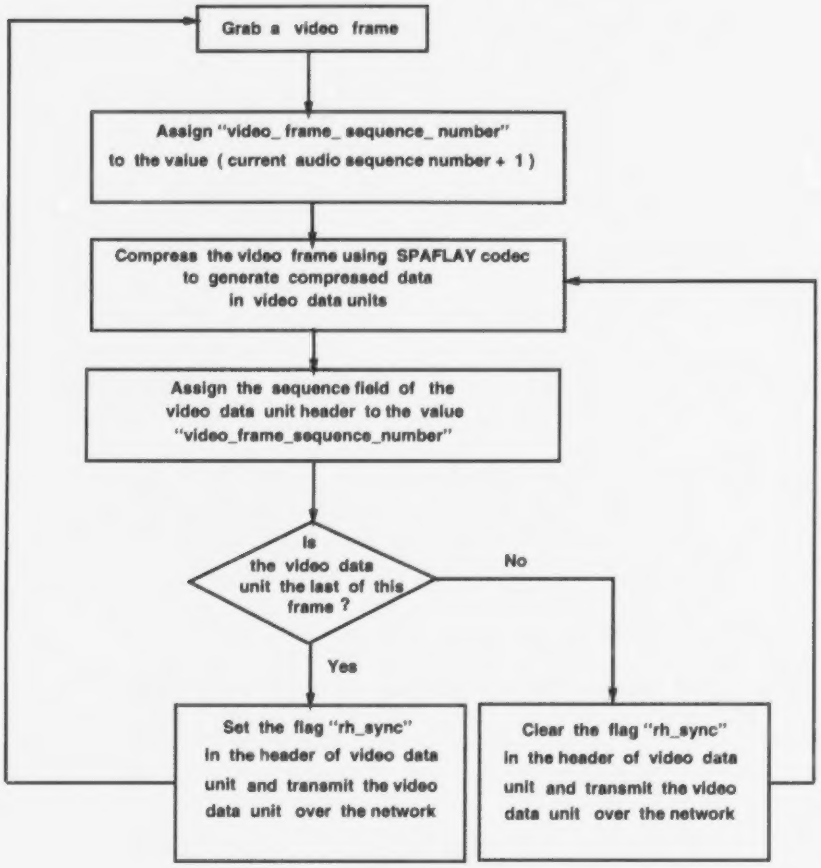

Grab a video frame
Assign "video_ frame_ sequence_ number" to the value (current audio sequence number + 1)
Compress the video frame using SPAFLAY codec to generate compressed data in video data units
Assign the sequence field of the video data unit header to the value "video_frame_sequence_number"
Is the video data unit the last of this frame ?
No
Yes
Set the flag "rh_sync" in the header of video data unit and transmit the video data unit over the network
Clear the flag "rh_sync" in the header of video data unit and transmit the video data unit over the network

Figure 62: Functionality of Sender-video-process

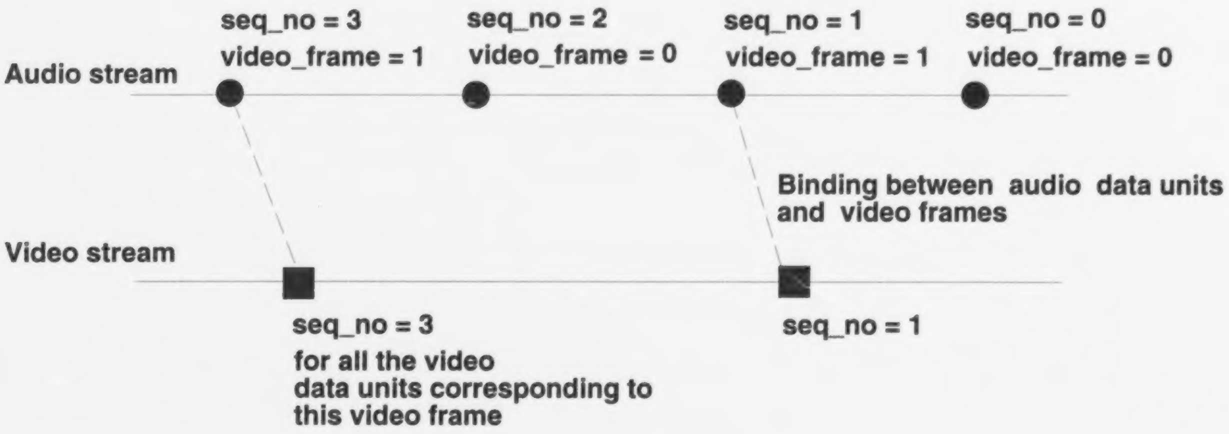

seq_no = 3
video_frame = 1
seq_no = 2
video_frame = 0
seq_no = 1
video_frame = 1
seq_no = 0
video_frame = 0
Audio stream
Binding between audio data units and video frames
Video stream
seq_no = 3
for all the video data units corresponding to this video frame
seq_no = 1

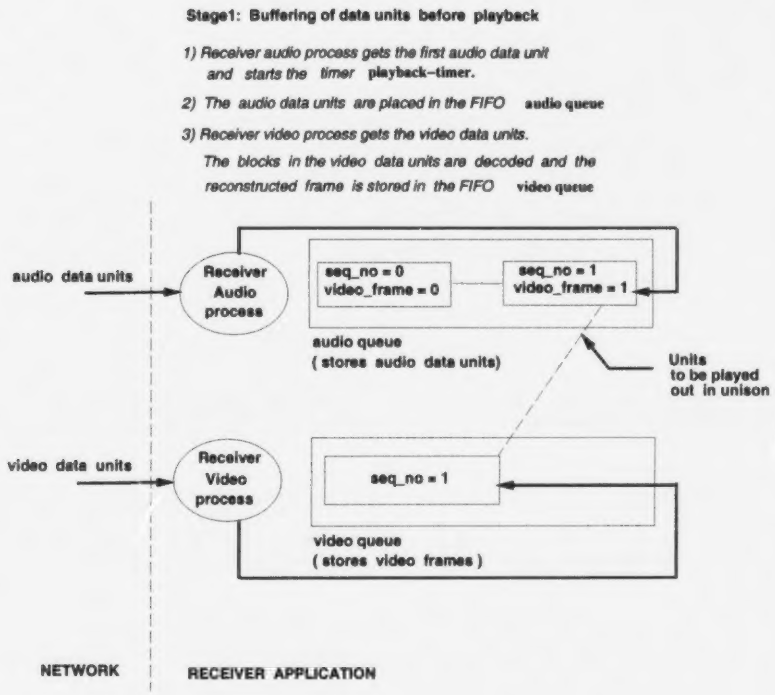
Figure 63: Binding audio data units with video frames

4.5.2 Temporal presentation control

Our scheme for presentation is also based on the **master-slave** model. In this model, the master stream (audio) controls the playback of slave medium (video). We use one of the intra-medium synchronization techniques to play out the samples of the master stream. Samples of the master stream contain information regarding the presence of slave medium samples which need to be played in unison with the master stream. The samples of the slave medium are played out as and when indicated by the samples of the master stream.

As indicated in our model of intra-medium synchronization, samples of a medium are buffered for a period equal to the maximum variable delay anticipated in the network before playback. Two separate FIFO queues are maintained to buffer audio data units and video frames. Here, it must be noted that audio data units are available directly to the receiver/client from the network. However, the same is not the case for video frames. As per our SPAFLAY coding algorithm, only *motion blocks* of each frame are transmitted. Hence, the receiver has to reconstruct the entire frame before buffering it in the video frame queue.

The receiver/client application has two separate modules to receive audio and video data units. The Client-Audio-Module receives audio data units and buffers them in the FIFO audio queue. The Client-Video-Module receives video data units, decodes them and stores the reconstructed frames in the FIFO video queue. When the audio module gets the first audio data unit, it is buffered in the audio queue. A timer named *playback-timer*, equivalent to the maximum variable delay, is also started with the receipt of the first audio data unit. When a video data unit is received, the receiver video process decodes the video blocks in this data unit. The decoded blocks are used to reconstruct the entire video frame. If the **rh_sync** bit in the header of a received video data unit is set, it is an indication to the receiver that the reconstructed frame is now complete. This frame is then stored in the FIFO video queue. The sequence number associated with each video data unit of this frame is also stored in the queue along with the video frame. The operations in this step are



Stage1: Buffering of data units before playback

1) Receiver audio process gets the first audio data unit and starts the timer **playback-timer.**

2) The audio data units are placed in the FIFO **audio queue**

3) Receiver video process gets the video data units. The blocks in the video data units are decoded and the reconstructed frame is stored in the FIFO **video queue**

audio data units
Receiver Audio process
seq_no = 0
video_frame = 0
seq_no = 1
video_frame = 1
audio queue
(stores audio data units)
Units to be played out in unison
video data units
Receiver Video process
seq_no = 1
video queue
(stores video frames)
NETWORK
RECEIVER APPLICATION

Figure 64: Playback: Stage 1

summarized in Fig. 64.

Playback begins when the *playback-timer* expires. Playback is handled by the *playbackhandler* process. The samples of the audio stream are removed from the FIFO audio queue and played out using one of the intra-medium synchronization techniques discussed earlier. It must be noted here that the *master* stream alone is played out by the *playbackhandler* process using the intra-medium synchronization technique. The header of the master medium playout units (audio) indicates whether there is a slave medium playout unit (video frame) bound to this master medium unit. If there is a slave playout unit associated with the current master medium playout unit, then the *playbackhandler* process plays out the slave medium unit along with the master medium playout unit.

Thus, playout of the slave medium is dependent on the master medium. There is a possibility of a slave medium unit bound to a master medium unit being absent. In the event of network congestion, a video frame could be lost. When an audio data unit bound to this video frame is played out, the *playbackhandler* process checks the FIFO video queue to see if the corresponding video frame is present. In the absence of the right video frame, the *playbackhandler* process plays the audio data unit alone. Thus the audio data units are played out irrespective of the presence of matching video frames. We must also note that as per this **master-slave** scheme, if a master medium data unit is lost and there exists a slave medium unit bound to this master medium unit, which has successfully reached the receiver, the slave medium unit will not be played out. Thus the master stream has complete control on the playback of slave medium units.

Playback of master medium units using Blind-Timing: Before playing out an audio data unit, the *playbackhandler* process checks the header of the audio data unit. If the **video_frame** flag is set, it indicates that there is a video frame which needs to be played out with this audio data unit. Accordingly, the *playbackhandler* process removes the video frame from the head of the FIFO video queue. A check is also made to determine whether the sequence number of the video frame matches the current audio data unit. If the sequence numbers match, the *playbackhandler* process displays the video frame and also plays out the audio data unit. The *playbackhandler* process then sleeps for a constant period. This constant period in the Blind-Timing scheme is equal to the time interval between the capture of successive samples of audio data units at the sender. The *playbackhandler* process wakes up after this period to play out the next audio data unit and repeats the procedure discussed above.

Playback of master medium units using Absolute-Timing: The sequence of steps is similar to the Blind-Timing case. The difference is only in the duration of time for which the *playbackhandler* process sleeps between playout of audio data units. In this scheme, the audio data unit header contains the exact duration of time between the capture of the previous audio data unit and the present one. The *playbackhandler* process removes an audio data unit for playback from the FIFO audio queue. It also looks at the header of next audio

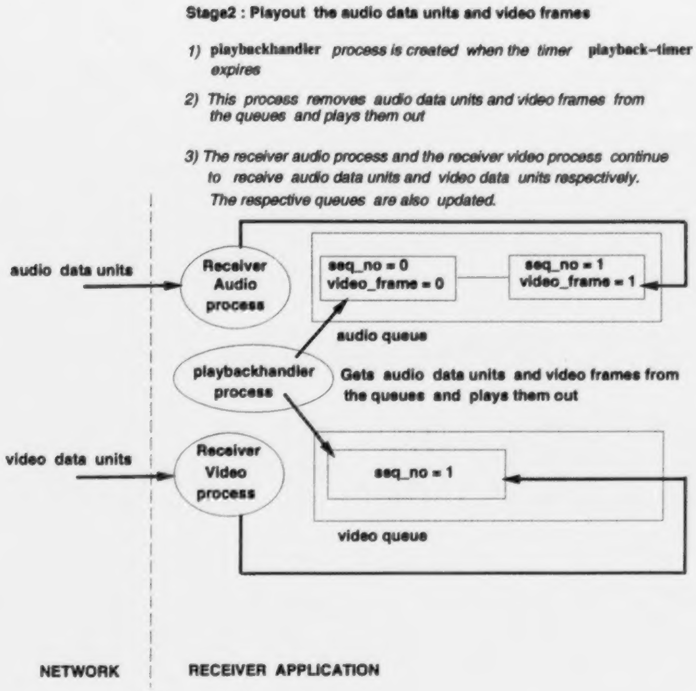

Stage2 : Playout the audio data units and video frames
1) playbackhandler process is created when the timer playback–timer expires
2) This process removes audio data units and video frames from the queues and plays them out
3) The receiver audio process and the receiver video process continue to receive audio data units and video data units respectively. The respective queues are also updated.
audio data units
Receiver Audio process
seq_no = 0
video_frame = 0
seq_no = 1
video_frame = 1
audio queue
playbackhandler process
Gets audio data units and video frames from the queues and plays them out
video data units
Receiver Video process
seq_no = 1
video queue
NETWORK
RECEIVER APPLICATION

Figure 65: Playback: Stage 2

data unit in the queue. From this, the process determines the exact duration of time for which it must sleep after playback of the current audio data unit. The *playbackhandler* process plays out the audio data unit with possibly a video frame as well and then sleeps for the prescribed duration. It wakes up to access the next audio data unit for playback and repeats the above procedure. The playback operations are depicted in Fig. 65.

The functionality of the receiver audio process is shown in Fig. 66. The receiver video process algorithm is shown in Fig. 67. The playbackhandler process can operate in two modes depending on whether Blind-Timing or Absolute-Timing is chosen as the intra-medium synchronization technique. When Blind-Timing is used, the playbackhandler process follows the algorithm presented in Fig. 68. When Absolute-Timing is used, the playbackhandler process follows the algorithm shown in Fig. 69.

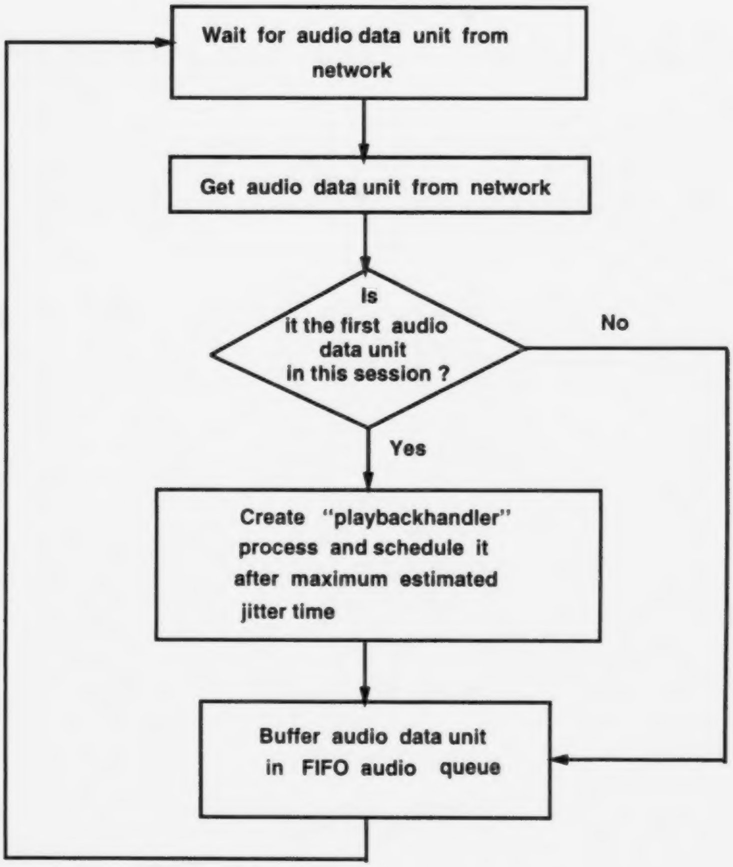

Wait for audio data unit from network
Get audio data unit from network
Is it the first audio data unit in this session ?
No
Yes
Create "playbackhandler" process and schedule it after maximum estimated jitter time
Buffer audio data unit in FIFO audio queue

Figure 66: Functionality of receiver audio process

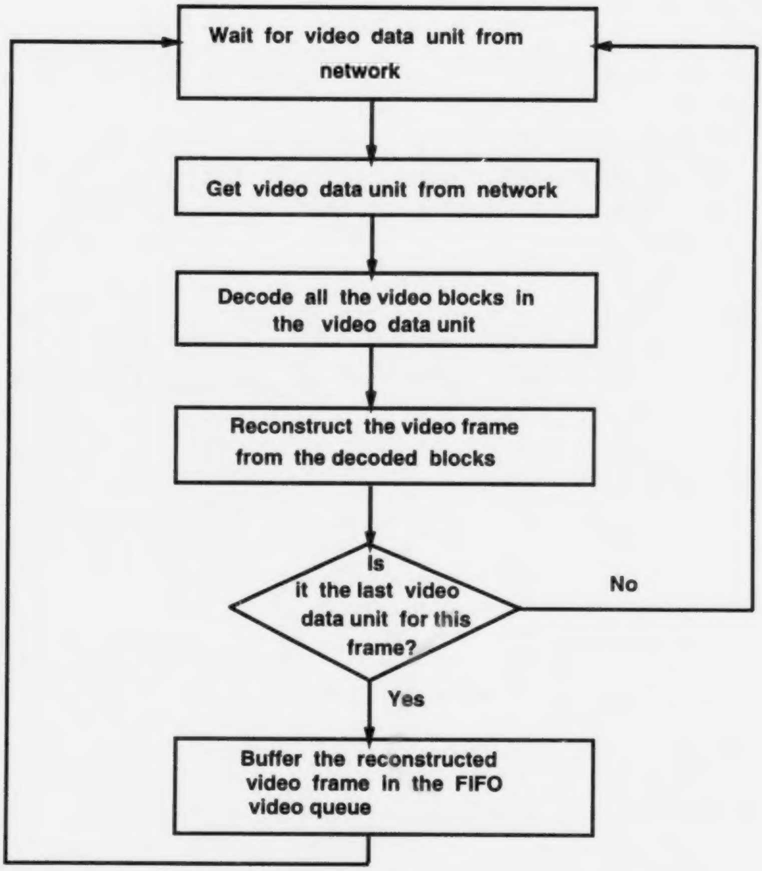

Wait for video data unit from network
Get video data unit from network
Decode all the video blocks in the video data unit
Reconstruct the video frame from the decoded blocks
Is it the last video data unit for this frame?
No
Yes
Buffer the reconstructed video frame in the FIFO video queue

Figure 67: Functionality of receiver video process

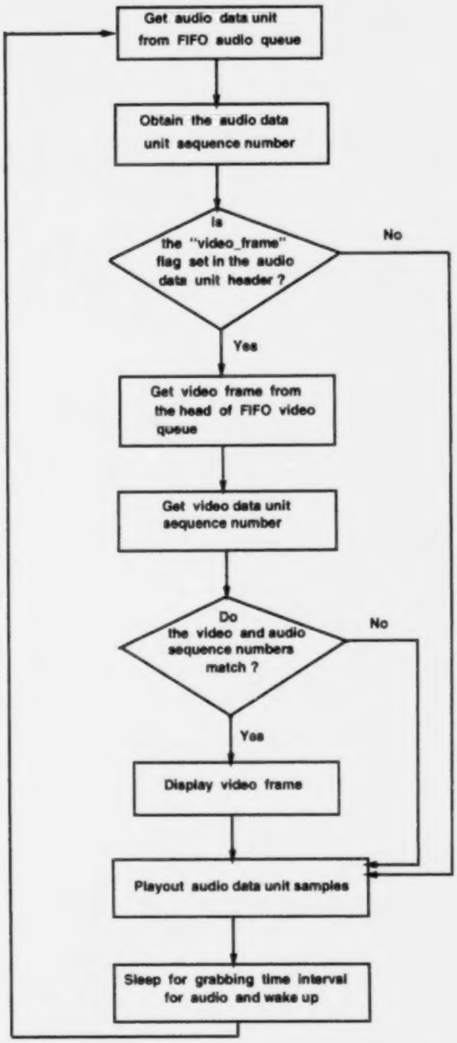

Get audio data unit from FIFO audio queue
Obtain the audio data unit sequence number
Is the "video_frame" flag set in the audio data unit header ?
No
Yes
Get video frame from the head of FIFO video queue
Get video data unit sequence number
Do the video and audio sequence numbers match ?
No
Yes
Display video frame
Playout audio data unit samples
Sleep for grabbing time interval for audio and wake up

Figure 68: Functionality of playbackhandler process (Blind-Timing)

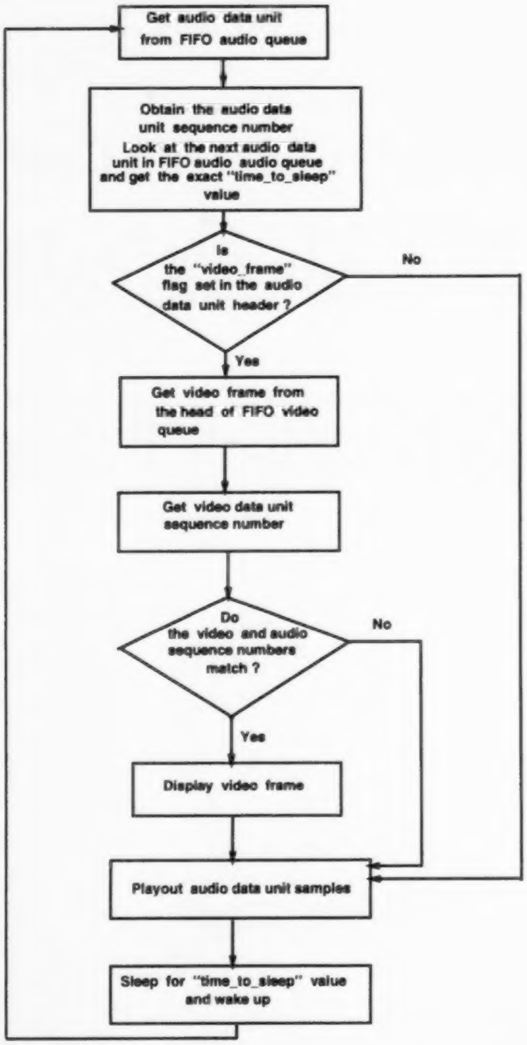

Get audio data unit from FIFO audio queue
Obtain the audio data unit sequence number
Look at the next audio data unit in FIFO audio audio queue and get the exact "time_to_sleep" value
Is the "video_frame" flag set in the audio data unit header ?
No
Yes
Get video frame from the head of FIFO video queue
Get video data unit sequence number
Do the video and audio sequence numbers match ?
No
Yes
Display video frame
Playout audio data unit samples
Sleep for "time_to_sleep" value and wake up

Figure 69: Functionality of playbackhandler process (Absolute-Timing)

4.6 Real-time Transport Protocol (RTP)

In the networked multimedia scenario, we have used RTP as the real-time protocol over UDP for transporting audio and video data. In this section, we provide an overview of RTP.

RTP is a transport protocol for real-time applications. It aims to provide services commonly required by interactive multimedia conferences, such as playout synchronization, media identification and active party identification. RTP itself does not provide any mechanism to ensure timely delivery or provide other quality-of-service guarantees, but relies on lower-layer services to do so. The sequence numbers included in RTP allow the end system to reconstruct the sender's packet sequence. RTP is designed to run on a variety of network and transport protocols, for example, IP, TCP and UDP.

RTP uses the services of an end-to-end transport protocol such as UDP and TCP. The services used are: end-to-end delivery, framing, demultiplexing and multicast. As an alternative, RTP could be used as a transport protocol layered directly on top of IP, potentially increasing performance and reducing header overhead. This may be attractive as the services provided by UDP, like calculating the checksum and demultiplexing, may not be needed for multicast real-time conferencing applications.

In this context, we need to look at two protocols:

- The Real-time Transport Protocol (RTP), for exchanging data that has real-time properties. The RTP header consists of a fixed-length portion plus optional control fields.

- The RTP Control Protocol (RTCP), for conveying information about the participants in an on-going session. RTCP consists of additional header options that may be ignored without affecting the ability to receive data correctly. RTCP is used for *loosely controlled* sessions, i.e., where there is no explicit membership control and set-up.

4.6.1 Use of RTP

In a multicast audio conference environment, RTP provides the following services. In an audio conferencing application, each conference participant sends audio data in small chunks of, say, 20 ms duration. Each chunk of audio data is preceded by an RTP header. The RTP header and data are in turn contained in a UDP packet. The Internet, akin to other packet networks, at times loses and reorders packets and delays them by a variable amount. To cope with these problems, the RTP header contains timing information and a sequence number that enable the receivers to reconstruct the timing as seen by the source, so that, in our example, a chunk of audio is delivered to the speaker every 20 ms. The sequence number can also be used by the receiver to estimate how many packets are being lost. Each RTP packet also indicates the audio encoding type being used, so that senders can change the encoding during a conference, to accommodate a new participant that is connected through a low-bandwidth link.

During the course of the conference, it might be useful to know the members who are participating at any moment. For this purpose, each instance of the audio application in the conference periodically multicasts the name, email address and other information of its user. This control information is carried as RTCP SDES (Real-time Control Protocol Source Descriptor) options within RTP messages, with or without audio data.

4.6.2 RTP Definitions

Payload is the data following the RTP fixed header and any RTP/RTCP options. Examples of payload are audio samples and video data.

RTP packet consists of the encapsulation specific to a particular underlying protocol, the fixed RTP header, RTP and RTCP options, if any, and the payload, if any.

(Protocol) port is an abstraction that transport protocols use to distinguish among multiple destinations within a given host computer. TCP/IP protocols identify ports using small positive integers.

Transport address denotes the combination of network address, e.g., the 4-octet IP version 4 address, and the transport protocol port, e.g., the UDP port. The destination transport address may be a unicast or multicast address.

Content source is the actual source of the data carried in an RTP packet, for example, the application that initially generated audio data.

Synchronization unit consists of one or more packets that are emitted contiguously by the sender. The most common synchronization units are talkspurts for voice and frames for video transmission. During playout synchronization, the receiver must reconstruct exactly the time difference between packets within a synchronization unit. In case of video, all the packets of a frame are given the same timestamp so there is no time difference. The time difference between synchronization units may be changed by the receiver to adjust to the network jitter.

4.6.3 RTP Fixed Header Fields

The first eight octets are present in every RTP packet. They represent the fixed header fields and have the following structure:

```
struct rtphdr
{
  uint8   rh_chanid:6;    /* channel id */
  uint8   rh_vers:2;      /* version */
  uint8   rh_content:6;   /* content id */
  uint8   rh_sync:1;      /* end of synchronization unit */
  uint8   rh_opts:1;      /* options present */
  uint16  rh_seq;         /* sequence number */
  uint32  rh_ts;          /* time stamp (middle of NTP timestamp) */
}
```

protocol version: 2 bits

Identifies the protocol version. The version number of the protocol defined is 1.

channel ID: 6 bits

The channel identifier field forms part of the tuple identifying a channel to provide an additional level of multiplexing at the RTP layer. The channel field is convenient if several different channels are to receive the same treatment by the underlying layers or if a profile allows for the concatenation of several RTP packets on different channels into a single packet of the underlying protocol layer.

option present bit (P): 1 bit

This flag has a value of one if the fixed RTP header is followed by one or more options and a value of zero otherwise.

end-of-synchronization-unit (S): 1 bit

This flag has a value of one in the last packet of a synchronization unit and a value of zero otherwise.

format: 6 bits

It identifies the RTP payload and determines its interpretation by the application.

sequence number: 16 bits

The sequence number counts the RTP packets. The sequence number increments by one for each packet sent. The sequence number may be used by the receiver to detect packet loss, to restore packet sequence and to identify packets to the application.

timestamp: 32 bits

The timestamp reflects the wall clock time when the RTP packet was generated. Several consecutive RTP packets may have equal timestamps if they were generated at once. The timestamp consists of the middle 32 bits of a 64-bit NTP timestamp. That is, it counts time since 0 hours UTC, January 1, 1900, with a resolution of 65536 per second. (UTC is Coordinated Universal Time, approximately equal to the historical Greenwich Mean Time.) The RTP timestamp wraps around approximately every 18 hours.

The RTP optional fields are currently not being used in the application.

Chapter 5

Application functionality

Advances in networking have made it feasible for digital computer networks to support multimedia communication. With advancement in the field of storage technologies, they can be used to build multimedia on-demand services over metropolitan-area-networks such as the Broadband Integrated Services Digital Network (B-ISDN), which are expected to permeate residential, organizational, and educational premises in a manner similar to existing telephone networks. In this chapter, we discuss the design and implementation of a multimedia server capable of servicing a number of on-demand retrieval requests. The server also supports multicasting multimedia information to a group of clients.

The application was developed to test the performance and effectiveness of the SPAFLAY video codec and synchronization techniques elaborated in the previous chapters. The application provides a range of features. Local recording, remote recording, interactive playback of orchestrated applications present locally and over a network are enabled by the application. It can also disseminate live audio and video information to a group of stations. We will study the functionality of the application under three main headers:

- Multimedia on-demand service
- Multimedia live multicast service

- Multimedia local playback service

5.1 Multimedia on-demand service

A multimedia on-demand server provides services similar to those of a neighbourhood videotape rental store. It digitally stores multimedia information consisting of audio and video data in secondary storage devices. Subscribers can connect to this server and request for transmitting the chosen media segments over the network to their respective sites. The retrieval can be interactive, in the sense that subscribers can stop, pause, resume, record, rewind and fastforward the media information. Thus, the multimedia server subsumes the function of VCRs, videotapes, audio recorders, etc., and can serve varied needs of clientele.

The development of a multimedia on-demand service is guided by three requirements of media playback:

Time continuum of each medium sample: A medium stream consists of a sequence of medium quanta such as video frames or audio samples, which convey meaning only when presented continuously in time. This is different from a textual object, where spatial continuity is sufficient. A multimedia server must ensure that playback of each medium proceeds at its real-time rate. Whereas ensuring the continuity of an isolated medium stream is relatively straightforward, a multimedia server's functionality is more complex since it has to cater to multiple subscriber requests without violating the playback rates of any of their requested media streams.

Synchronization between media streams: Playback of multiple media streams constituting a multimedia object should not only be continuous but also temporally coordinated. In the event of jitter delays in the network, the playback of media streams can go out of synchronization.

Different subscriber states: Unlike a video conferencing session where the same information is sent to all subscribers, a multimedia on-demand service is characterized by the server having to cater to diverse requests from clientele. Subscribers could connect to the multimedia server with requests for different multimedia segments. Even within the same segment, each client could potentially be in a different state with respect to other clients due to the availability of interactive retrieval facility. A client could rewind, fastforward and pause within a multimedia segment. Hence, the multimedia server cannot multicast the same information to all the clients because the media information could be different for each client. The server must receive commands from each subscriber and respond to these requests individually.

In the next section, we discuss the on-demand information service functionality provided by our application.

5.1.1 On-demand service system architecture

The system architecture of a multimedia on-demand service comprises of a multimedia server. Clients can connect to this server via a network (LAN/MAN). The multimedia server needs to have access to storage disks, while the clients must have simple media capture-and-display subsystems (e.g., camera, speakers). The system architecture will be presented considering the client and the server separately. We will also look into the protocols used to transfer information between the client and the server.

Client architecture: We begin this description of the client architecture by looking at the various modules [30] which constitute the client. The client entity consists of four modules:

- Client-Control-Module
- Client-Audio-Module
- Client-Video-Module
- Client-Playback-Module

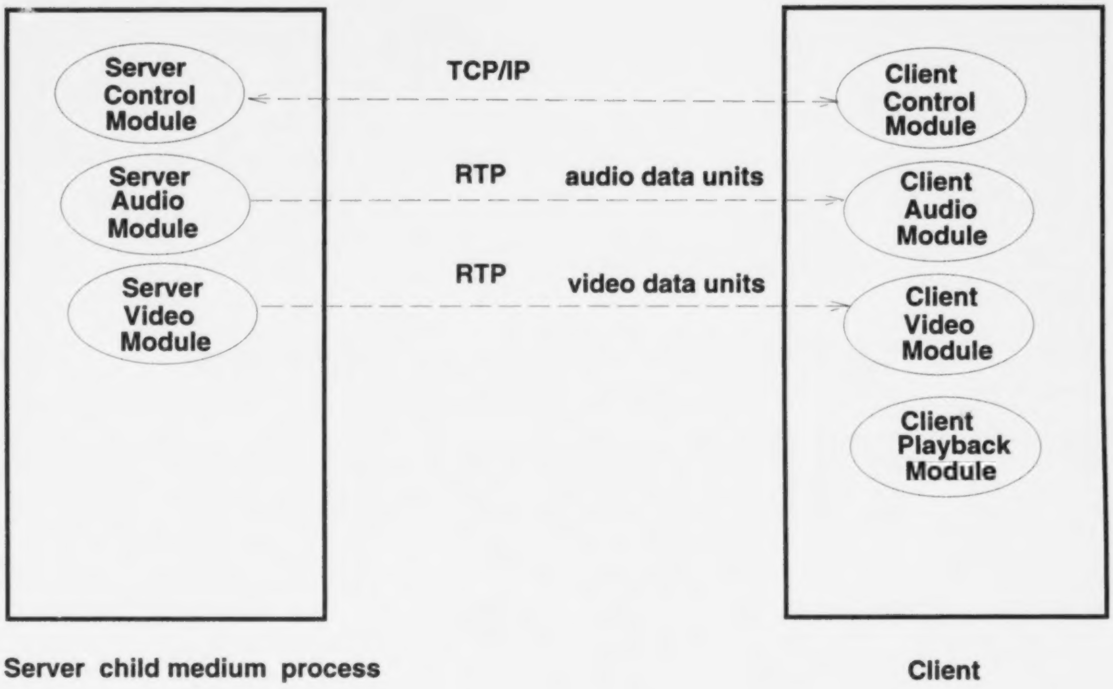

Server Control Module
Server Audio Module
Server Video Module
TCP/IP
RTP
audio data units
RTP
video data units
Client Control Module
Client Audio Module
Client Video Module
Client Playback Module
Server child medium process
Client

Figure 70: Basic modules in the client and the server

These client modules are shown in Fig. 70.

Client-Control-Module: The user interacts with the Client-Control-Module via the user-interface panel. We look at the various user options available to access and interact with the on-demand service provided by the remote multimedia server. When the options are selected by the user, the Client-Control-Module sends these client requests as control messages to the server. These messages are sent using a TCP/IP connection to the multimedia server. Likewise, control messages sent by the server to the client are also received by this module. The control messages are sent by the client to the server to communicate subscriber retrieval requests. The TCP/IP protocol suite is used over the control channel to send messages reliably to the server.

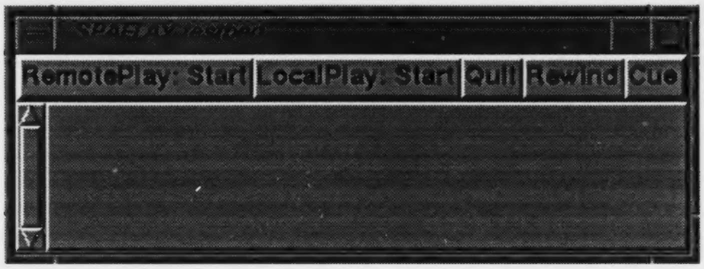


Figure 71: Client panel

The client user-interface for the multimedia session is shown in Fig. 71. The user is presented with the following options:

- Remoteplay:Start
- Localplay:Start
- Rewind
- Cue
- Quit

To establish a multimedia on-demand session with a remote server, the client must choose the option **Remoteplay:Start**. When the **Remoteplay:Start** option is selected, the user is prompted to input the address of the server as also the parameters to configure the multimedia session. Specifically, the user must enter values for the following list of options:

- **server-name:** The multimedia server machine name.
- **server-port-number:** The port which the multimedia server advertises to clients which may want to connect to it.
- **record-enable:** The user is given the option to record the contents of the present session. If the user wishes to record the session, he is prompted to input the name of the audio file and the video file in which compressed media data will be stored.
- **intra-medium-synchronization-protocol:** There are two intra-medium synchronization techniques which the user may use. An input of 0 sig-

nifies the Blind-Timing scheme, while an input of 1 means Absolute-Timing.

- **orchestrated-application:** An input of 1 signifies request for playback of an orchestrated application. The user is prompted to input the name of the media streams that he wishes to play. This is typically made up of an audio file and a video file.

The following control messages are sent by the Client-Control-Module to the server:

Connect-request: This message is sent when the **RemotePlay:Start** option is selected by the client. When the **RemotePlay:Start** button is pressed at the start of the session, it is a request by the client to connect to a remote multimedia server to access the on-demand service. The subscriber selects options for this playback session as indicated earlier. These options are sent to the server in the **Connect-request** message.

Session-pause: When a **Connect-request** message is sent by selecting the **RemotePlay:Start** option, the multimedia session is assumed to have begun. The **RemotePlay:Start** option is replaced by the option **RemotePlay:Stop**. If the user presses this button during the course of the multimedia session, it is an indication to the server to pause. In this case, a **Session-pause** control message is sent by the client to the server. The multimedia session is stopped for the time being. The option **RemotePlay:Stop** is again replaced by **RemotePlay:Start**. Selecting this option now results in the **Session-Resume** message being sent to the server. The playback session is resumed.

Connect-release: During the playout of a multimedia session, the user may choose to terminate the session. The user selects the **Quit** option to abort the session. When this option is selected, a **Connect-release** message is sent by the Client-Control-Module to the server. The server discontinues the on-going multimedia session.

Session-fastforward: An interactive orchestrated application should enable the user to scroll through the presented multimedia material. Specifically, the user should be able to playback multimedia material from a point which is ahead in time to the material which is currently being played. The user selects the **Cue** button to enable this feature. The Client-Control-Module sends the **Session-fastforward** message to the server. Both the audio and video streams are synchronized to a point which is ahead in time by the server. The extent of this displacement of the present session to a future point in time is configurable by the user. It is given by the parameter *fastforward-stepsize.* The parameter *fastfoward-stepsize* is added to the sequence number of the audio data unit which was played out last. The resultant audio data unit sequence number represents the point in the audio stream from where the session will again commence after the **Cue** button has been released. The server must synchronize the video stream to coincide with this point in the audio stream.

Session-rewind: Akin to the fastforward feature presented earlier, we also have a session rewind option. The user presses the **Rewind** button to playback session (audio-video) material from an earlier point in time. The extent of the displacement of the session material in time is expressed in terms of the parameter *rewind-stepsize.* The parameter *rewind-stepsize* is subtracted from the sequence number of the last played out audio data unit. The resultant audio data unit sequence number represents the point in the audio stream from where the session will again commence, after the **Rewind** button has been released. The server must also synchronize the video stream to coincide with this point in the audio stream.

Client-Audio-Module: The Client-Audio-Module receives audio data units from the server. The audio data units are received as RTP (Real-time Transport Protocol) packets by the module. We have used RTP as the real time protocol over UDP for transporting audio and video data. The audio data units are buffered in a FIFO audio queue. As was explained in **Chapter 4**, the first audio data unit is buffered for a time equal to the maximum variable delay over the network. The audio data units are removed from the head of the audio queue by the Client-Playback-Module and played out.

Client-Video-Module: The Client-Video-Module receives video data units from the server. The video data units are received as RTP (Real-time Transport Protocol) packets. The video blocks in the video data units are decoded using the SPAFLAY codec and a frame is reconstructed. When the end-of-frame marker is set in the header of a video data unit, the reconstruction of the frame is complete. The decoded frame is stored in a FIFO video queue. The frames are picked up by the Client-Playback-Module and played out.

Client-Playback-Module: The Client-Playback-Module is responsible for the synchronized playback of audio and video at the client's end. The module picks up audio and video data from head of the audio and video queues respectively and plays them out. The Client-Playback-Module uses one of the two modes of intra-medium synchronization techniques: Blind-Timing and Absolute-Timing. Selection of the mode depends on the value of the *intra-medium-synchronization-protocol* parameter input by the user at the start of the multimedia session. Depending on selection of the intra-medium synchronization technique, the Client-Playback-Module schedules itself to remove audio data units from the head of the FIFO audio queue. The audio data units are decoded and played out. Video frames are displayed in accordance with the **master-slave** inter-media synchronization

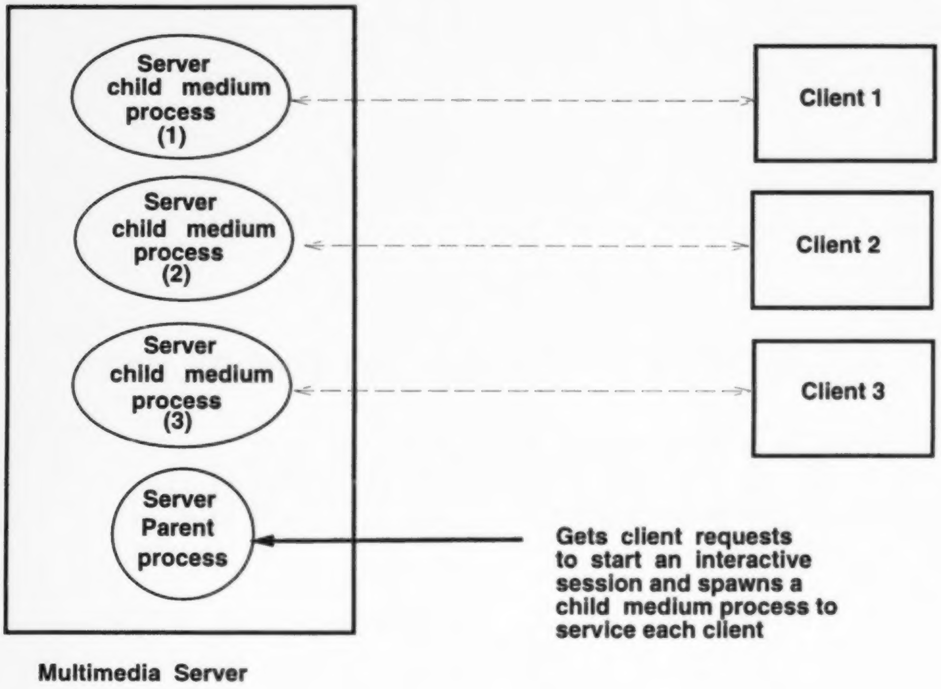

Server child medium process (1)
Server child medium process (2)
Server child medium process (3)
Server Parent process
Client 1
Client 2
Client 3
Gets client requests to start an interactive session and spawns a child medium process to service each client
Multimedia Server

Figure 72: Orchestrated playback for multiple clients

technique elucidated in **Chapter 4**.

Server architecture: Different subscribers may request retrieval of different media streams. There may also be phase differences among their requests even if the request be for different portions of the same medium stream. Hence, the possibility of multicasting information from the server to the client is ruled out. The server creates a separate process to handle each client request. The server receives the **Connect-request** message and spawns a child medium process. The child medium process services the client requests throughout the session, enabling the parent process to listen to other subscriber requests. We illustrate this in Fig. 72. Each child medium process has three distinct modules:

- Server-Control-Module
- Server-Audio-Module
- Server-Video-Module

It is illustrated in Fig. 70.

Server-Control-Module: The Server-Control-Module receives control messages from the client. The action taken on receipt of each control message sent by the Client-Control-Module is as follows:

- **Connect-release:** When the Server-Control-Module receives this message, it is a request from the client to terminate the multimedia session. The Server-Control-Module accordingly terminates the process servicing the client.
- **Session-pause:** Receipt of this message is an indication to the server to temporarily stop sending audio and video data to the client till a **Session-resume** message is again received from the client. The Server-Control-Module sends a message to the Server-Audio-Module and Server-Video-Module to suspend their functions.
- **Session-resume:** The multimedia session which had been temporarily stopped due to the receipt of a **Session-pause** message is restarted when a **Session-resume** message is received. The Server-Control-Module sends a message to the Server-Audio-Module and Server-Video-Module to start sending audio and video data units again.
- **Session-fastforward:** The message is a request by the client to playback audio and video information from a point which is ahead in time to the information which is currently played at the client's end.

  The **Session-fastforward** message contains as one of its fields, the audio sequence number to be played out next. On receiving this message, the Server-Control-Module sends a message to the Server-Audio-Module and Server-Video-Module to suspend their functions. The control module at the sender, the Server-Control-Module, thereafter adjusts the pointer in the audio file to point to the audio data unit with sequence number equal to the value specified in the message.

The video file pointer is also changed to synchronize it with the point in the audio stream. It means changing the video file pointer to the video data unit with a sequence number equal to the value specified in the message. The Server-Control-Module then sends a message to the Server-Audio-Module and Server-Video-Module to resume their functions. The modules start sending audio and video data units from the new point in their media streams.

- **Session-rewind:** The message is a request by the client to playback audio and video information from a point in time which is prior to the information which is currently played at the client's end.

  The **Session-rewind** message contains as one of its fields, the audio sequence number to be played out next. On receiving this message, the Server-Control-Module sends a message to the Server-Audio-Module and Server-Video-Module to suspend their functions. The Server-Control-Module thereafter adjusts the pointer in the audio file to point to the audio data unit with sequence number equal to the value specified in the message. The video file pointer is also changed to synchronize it with the point in the audio stream. It means changing the video file pointer to the video data unit with a sequence number equal to that specified in the message. The Server-Control-Module then sends a message to the Server-Audio-Module and Server-Video-Module to resume their functions. The modules start sending audio and video data units from the new point in their media streams.

Server-Audio-Module: At the start of the multimedia session, the user specifies the name of the audio file which contains audio data to be played out. The Server-Control-Module gets this specification in the **Connect-request** message. It is passed to the Server-Audio-Module.

For playback of an orchestrated application, the Server-Audio-Module has to read audio data units repeatedly from the audio file. The module sends these audio data units using the Real-time Transport Protocol to the client.

Server-Video-Module: At the start of the multimedia session, the user also specifies the name of the video file which contains video data to be played out. The Server-Control-Module gets this specification in the **Connect-request** message. It is passed to the Server-Video-Module.

For playback of an orchestrated application, the Server-Video-Module has to read video data units repeatedly from the video file. The video data units contain video data in a compressed form. The video data units are sent using the Real-time Transport Protocol to the client.

5.2 Multimedia live multicast service

Orchestrated applications which need to be played back in an interactive manner to the clients cannot be serviced by a server operating in a multicast mode. The clients could potentially be in various states playing out material from different parts of the same data stream. Hence, at the expense of inefficient multimedia information distribution, we resort to unicasting multimedia information individually to the clients.

On the other hand, some applications demand multimedia information to be multicast to a group of stations. A conference session might have to be relayed to people sitting at geographically separated sites. In this case, there is little interaction between the client and the server multicasting this information, restricted only to clients joining or leaving the session. In this case, we could resort to multicasting.

A multicast service can offer two benefits [11] to network applications :

Efficient multi-destination delivery: When an application must send the same information to more than one destination, multicasting is more efficient than *unicasting* separate copies to each destination. It reduces the transmission overhead at the server and, depending on how it is implemented, it can reduce the overhead on the network and the time taken for all destinations to receive the information. Examples of applications that can take advantage of multi-destination delivery are:

- updating all copies of a replicated file or database.
- sending voice, video, or data packets to all participants in a computer mediated conference.
- disseminating intermediate results to a set of processors supporting a distributed computation.

Robust unknown-destination delivery: If a set of stations can be identified by a single *group address* (rather than a list of individual addresses), such a group address can be used to reach one or more destinations whose individual addresses are unknown to the sender, or whose addresses may change over time. Sometimes called *logical addressing* or *location-independent addressing*, the use of multicast serves as a simple, robust alternative to configuration files, directory servers, or other binding mechanisms.

Because of these benefits, multicasting has seen widespread use in those networks that support it, primarily local-area networks. Hence, we provide an option for the multimedia server to operate in a multicast mode. Accordingly, our architecture for the client-server combine undergoes a little change when operating in the multicast mode. It is illustrated in Fig. 73. We use IP multicast datagrams to disseminate audio-video data to all the clients.

5.2.1 Sending IP multicast datagrams

To send a multicast datagram [10], an IP address is specified in the range 224.0.0.0 to 239.255.255.255. This destination address is used in a sendto() call.

By default, IP multicast datagrams are sent with a time-to-live (TTL) of 1 (see below), which prevents them from being forwarded beyond a single subnetwork. A new socket option allows the TTL for subsequent multicast datagrams to be set to any value from 0 to 255, in order to control the scope of the multicasts:

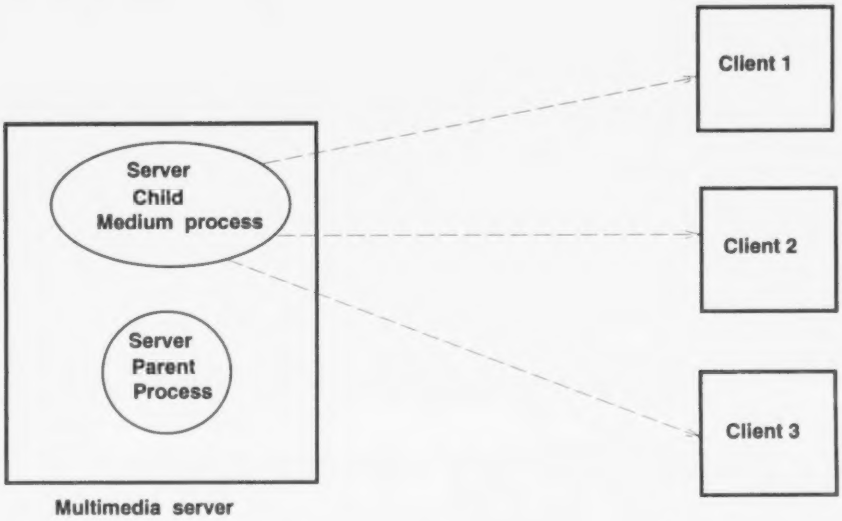


Figure 73: Multimedia multicast service for multiple clients

```
u_char ttl;
setsockopt(sock, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(ttl));
```

Multicast datagrams with a TTL value of 0 will not be transmitted on any subnet, but may be delivered locally if the sending host belongs to the destination group and if the multicast loopback has not been disabled on the sending socket. Multicast datagrams with a TTL greater than one may be delivered to more than one subnet if there are one or more multicast routers attached to the first-hop subnet. To provide meaningful scope control, the multicast routers support the notion of *thresholds* which prevents the datagrams with a certain TTL from traversing certain subnets. The thresholds enforce the following convention:

- multicast datagrams with initial TTL 0 are restricted to the same host.
- multicast datagrams with initial TTL 1 are restricted to the same subnet.
- multicast datagrams with initial TTL 32 are restricted to the same site.

- multicast datagrams with initial TTL 64 are restricted to the same region.
- multicast datagrams with initial TTL 128 are restricted to the same continent.
- multicast datagrams with initial TTL 255 are unrestricted in scope.

Sites and *Regions* are not strictly defined, and sites may be further subdivided into smaller administrative units, as a local matter.

5.2.2 Receiving IP multicast datagrams

Before a host can receive IP multicast datagrams, it must become a member of one or more IP multicast groups. A process can ask the host to join a multicast group by using the following socket option:

```
struct ip_mreq mreq;
setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq));
```

where **mreq** is the following structure:

```
struct ip_mreq {
struct in_addr imr_multiaddr; /* multicast group to join */
struct in_addr imr_interface; /* interface to join on */
}
```

Every membership is associated with a single interface, and it is possible to join the same group or more than one interface. The field **imr_interface** should be INADDR_ANY to choose the default multicast interface, or one of the host's local addresses to choose a particular (multicast-capable) interface.

To drop a membership, we use:

```
struct ip_mreq mreq;
setsockopt(sock, IPPROTO_IP, IP_DROP_MEMBERSHIP, &mreq, sizeof(mreq))
```

where **mreq** contains the same values as used to add a membership. The memberships associated with a socket are also dropped when a socket is closed or when a socket is killed.

5.2.3 Client options

The client panel for the live, multicast, multimedia session is the same as that shown in Fig. 71. Among the options presented to the user, **Remoteplay:Start** and **Quit** alone assume significance for a multicast service. The options: **Rewind**, **Cue** and **LocalPlay:Start** have no significance in a multicast session which is not an interactive service. The user has control only to join or leave a multimedia session.

To join an audio-video multicast session, the user must choose the option **Remoteplay:Start**. When this option is selected, the user is prompted to input values for parameters to be used in the present session. The user must enter values for the following list of options:

- **server-name:** The multimedia server machine name.
- **server-port-number:** The port which the multimedia server advertises to clients which may want to connect to it.
- **record-enable:** The user is given the option to record the contents of the present session. If the user wishes to record the session, he is prompted to input the name of the audio file and the video file in which compressed media data will be stored.
- **orchestrated-application:** An input of 0 indicates request for a live multimedia session. The user has to select a *multicast* or *unicast* option for this live session. The user has to choose the *multicast* option to be in multicast mode.
- **intra-medium-synchronization-protocol:** There are two intra-medium synchronization techniques which the user may use. An input of 0 signifies the Blind-Timing scheme, while an input of 1 means Absolute-Timing.

These options are sent in a **Connect-request** message over a control channel to the server using the TCP/IP protocol suite to the server.

5.2.4 Client-server architecture

The Client modules remain the same as in the **Multimedia on-demand service**:

- Client-Control-Module
- Client-Audio-Module
- Client-Video-Module
- Client-Playback-Module

Their functions are almost the same as in the earlier service. The Client-Control-Module is used only to join and leave the multicast session. The Client-Audio-Module is concerned with the task of initializing the host to receive IP multicast audio datagrams. The received audio datagrams are buffered in a FIFO audio queue. The Client-Video-Module must also perform initializations to receive IP multicast video datagrams. The received video data units are used to reconstruct the video frames which are stored in a FIFO video queue. The Client-Playback-Module removes audio data units and video data units from the head of the queues and plays them out in keeping with the intra-medium synchronization technique and the master-slave inter-media playback technique.

The server receives the **Connect-request** message from each client. One child medium process alone caters to multicasting audio-video data to all clients. The modules in the child medium process remain the same as in the on-demand service case. Their functionality is slightly different for a multicast session.

Server-Control-Module: For a multicast session, the Server-Control-Module has no messages to receive from the clients.

Server-Audio-Module: The Server-Audio-Module grabs audio samples from the microphone in units of audio data units. The audio data units are multicast over the subnet as multicast datagrams with appropriate TTL value.

Server-Video-Module: The Server-Video-Module grabs video frames using the video camera mounted on the multimedia server. The video frames are compressed using the SPAFLAY codec and the resulting video data units are sent in multicast datagrams with appropriate TTL value.

5.3 Multimedia local playback service

While the earlier two service options were for a networked multimedia service, this service caters to playback of an orchestrated application which is present on the same site or workstation. Hence, there is no client-server paradigm while operating in this mode.

5.3.1 Client options

The options presented to the user are the same as in Fig. 71. The following options are significant for a local playback service.

- Localplay:Start
- Rewind
- Cue
- Quit

Localplay:Start: To start a local playback service, the user must choose the option **Localplay:Start**. When the option is chosen, the user is prompted to input values for parameters which are essential for this session. Specifically, the user must input values for the following options :

- **audio-file:** The name of the audio file which the user wishes to play out.
- **video-file:** The video file which has to be played out with the audio file.

- **intra-medium-synchronization-protocol:** There are two intra-medium synchronization techniques which the user may use. An input of 0 signifies the Blind-Timing scheme, while an input of 1 means Absolute-Timing.

On selecting **Localplay:Start** button, it is replaced by **Localplay:Stop** option. During the course of the session, the user can cause playback to stop temporarily by choosing the option **Localplay:Stop**. The button is used as a toggle to start and pause playback of the session.

Rewind: The user presses the **Rewind** button to playback session material from an earlier point in time.

Cue: The user chooses this option to playback session material from a point which is ahead in time to the material which is currently being played out.

Quit: The user chooses this option to terminate the session.

5.3.2 Playback modules

There are four modules to facilitate local playback:

- Local-Control-Module
- Local-Audio-Module
- Local-Video-Module
- Local-Playback-Module

Local-Control-Module: The Local-Control-Module sends messages to the audio and video modules and exercises control over the session. These messages are issued when the user chooses one of the panel options. When the **Localplay:Start** option is selected and the parameter values are input, the Local-Control-Module opens the media files for playback. If the **Localplay:Stop** option is chosen, the Local-Control-Module sends a message to the audio and video modules to suspend their functions. This causes playback to stop temporarily.

When the user selects the **Rewind** option, the Local-Control-Module sends a message to the audio and video modules to suspend their operation. The Local-Control-Module adjusts the audio and video pointers to a point which is earlier in time in the media streams. The audio and video modules thereafter resume their function.

If the **Cue** option is selected, the control module sends a message to the audio and video modules to suspend their functions. The control module adjusts the audio and video file pointers to a point which is ahead in time to the material which is currently being played. The control module sends messages to the audio and video modules to resume their function.

Local-Audio-Module: The Local-Audio-Module reads audio data units repeatedly from the audio file. The audio data units are buffered in a FIFO audio queue.

Local-Video-Module: The Local-Video-Module reads video data units from the video file. The video blocks in the video data units are decoded and the reconstructed frames are stored in a FIFO video queue.

Local-Playback-Module: The Local-Playback-Module removes audio and video data from the queues and plays them out in accordance with the intra-medium synchronization technique chosen and the master-slave inter-media synchronization technique.

5.4 Recording a multimedia session

An orchestrated multimedia session is stored as a series of *playback units.* Each playback unit is made up of an audio and video file. Dividing a session into a series of units is necessitated due to two reasons:

- Sequence numbers used to identify audio and video data units have a wraparound. If medium information is stored in a single file, then due to the presence of a wraparound in the sequence numbers, there could be a conflict in uniquely identifying a playback point on the basis of the sequence numbers in the media files. It is especially true when a **Rewind** or **Cue** option is chosen.
- By dividing a session into a series of units, the multimedia session is stored in a structured manner. It is possible for the user to identify a particular unit and start playback of the session from that point.

The user is not concerned with the process of recording a multimedia session into separate playback units. At the start of the session, the user specifies the name of an audio and video file in which media information is to be recorded. An index value of zero is appended to the name of the audio and video files. For example, if the user specifies *record-audio-file* and *record-video-file* as the names of the audio and video files in which the data is to be recorded, then the resultant file names become *record-audio-file0* and *record-video-file0*. This combination of an audio and video file constitutes the first playback unit. The files contain audio and video data respectively, with no wraparound in the sequence numbers identifying the data units.

Sequence numbers are two bytes long. This gives a range of values from 0 to 65535. Since the audio stream is the master medium stream (**Chapter 4**), the sequence numbers will wraparound after about 40 minutes. When there is a wraparound in the sequence numbers of a medium stream, the corresponding medium file is closed. The index value is incremented and is appended to the name of the medium file name specified initially. For the above example, the second playback unit is made up of the files *record-audio-file1* and *record-video-file1*. Thus each time there is a wraparound in the sequence numbers, a new playback unit is created.

Thus, a multimedia session is recorded in a series of units, each of which is identified by an index value which is appended to the names of the initially specified files. The concept of playback units is illustrated in Fig. 74.

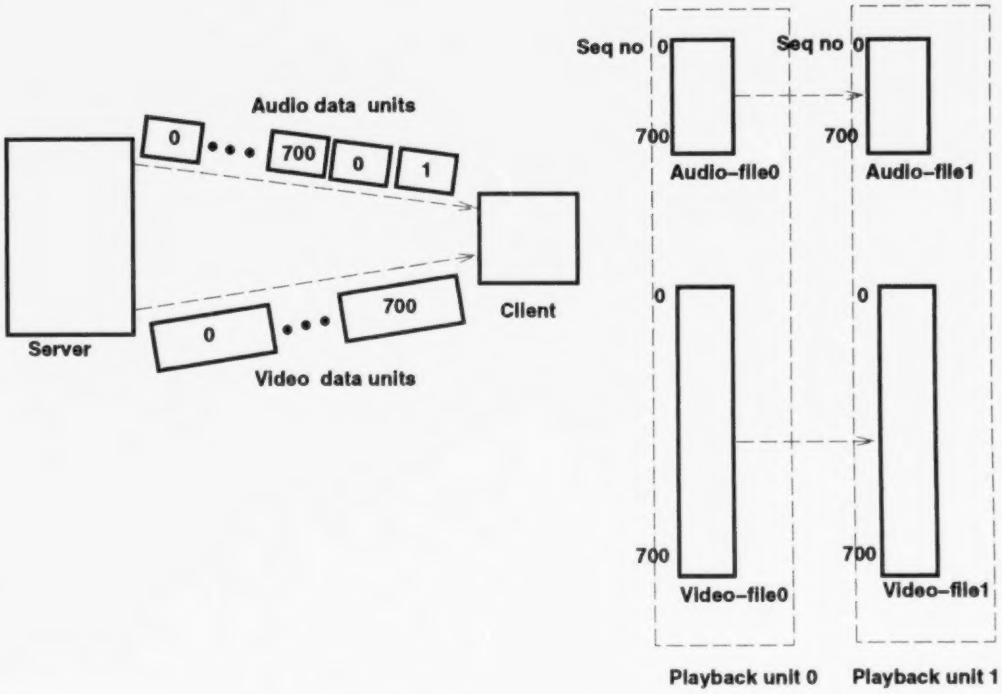

Audio data units
0
700
0
1
700
0
Video data units
Server
Client
Seq no
0
700
Audio-file0
Seq no
0
700
Audio-file1
0
700
Video-file0
0
700
Video-file1
Playback unit 0
Playback unit 1

Figure 74: Recording a multimedia session

5.5 Rewind and fastforward feature implementation

Rewind and fastforward requests are used to playback orchestrated material from a different point in time. This feature is provided in both networked and local playback of orchestrated applications. In our application, rewind and fastforward are used to move within the same playback unit, as also between playback units.

Providing Rewind and Fastforward features in two tiers helps to achieve *fine* and *coarse* scrolling within the recorded media material. By fine scrolling, we mean moving within the same playback unit. Using coarse scrolling, we can move between playback units.

In keeping with the master-slave inter-media synchronization technique, where the audio stream is the *master*, the rewind and fastforward feature also involve adjustment of the playback pointer in the audio stream. The position of the video stream playback pointer is thereafter changed to synchronize it with the audio stream.

The rewind and fastforward features depend on the parameters *rewind-stepsize* and *fastforward-stepsize* respectively. These parameters give the magnitude of the displacement (in terms of audio data unit sequence numbers) from the audio data unit which was played out last, when the rewind or fastforward option is chosen.

5.5.1 The Rewind feature

Let the sequence number of the last played out audio data unit in a playback unit be a. This sequence number represents the *playback point* in the playback unit. If a rewind option is selected now, we subtract the value *rewind-stepsize* from a.

If the resultant sequence value is greater than zero, then it represents a valid audio data unit in the playback unit. The audio file pointer is moved to the data unit with this sequence number. This sequence number becomes the new playback point. The video file pointer is also moved to a video data unit with the same sequence number. This is fine scrolling as applied to the rewind feature. For example, let *audio-file0* and *video-file0* constitute the the present playback unit. Let the sequence number 560 be the playback point. It means that the last audio data unit which was played out had a sequence number of 560. In terms of time units, this corresponds to 21 seconds of audio playback time from the start of the current playback unit. Let the *rewind-stepsize* parameter have a value of 400. In terms of time units, this corresponds to 15 seconds of audio playback time. If a rewind option is selected at this stage, we will shift to a playback point in the audio stream which was 15 seconds prior to the information which was played out last. To do this, we subtract the *rewind-step-size* value of 400 from 560. The resultant value, equal to 160, represents a valid sequence number in the playback unit. Both the audio and video file pointers are shifted to coincide with a data unit with this sequence number of 160. The sequence number 160 becomes the new playback point. It corresponds to 6 seconds of audio playback time from the start of the current playback unit. In effect, we have rewound the media streams to a playback point which was 15 seconds prior to the last samples

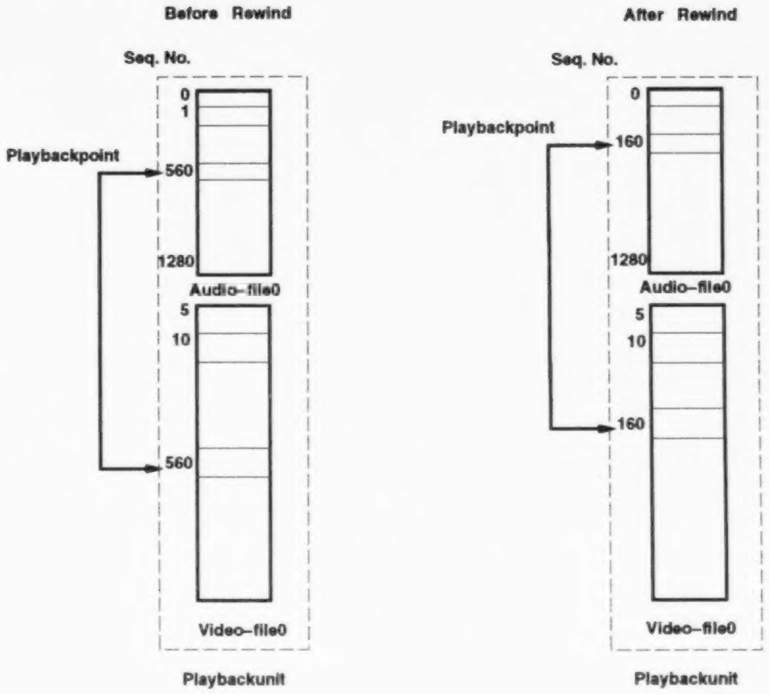

Before Rewind
Seq. No.
0
1
Playbackpoint
560
1280
Audio-file0
5
10
560
Video-file0
Playbackunit
After Rewind
Seq. No.
0
Playbackpoint
160
1280
Audio-file0
5
10
160
Video-file0
Playbackunit

Figure 75: Rewinding in a multimedia session

played out. This is illustrated in Fig. 75.

On the other hand, if the resultant value after the subtraction of sequence numbers is less than zero, it means that we wish to rewind to a point which is earlier than the first playback point in the present playback unit. Here, we resort to coarse scrolling. The current playback unit is skipped and we move to the start of the previous playback unit. For example, let *audio-file1* and *video-file1* constitute the present playback unit. Let the sequence number 240 be the playback point. It means that the last audio data unit which was played out had a sequence number of 240. Expressed in time units, it corresponds to 9 seconds of audio playback time from the start of the current playback unit. Let the *rewind-stepsize* parameter have a value of 400. It corresponds to 15 seconds of audio playback time. If a rewind option is selected at this stage, it means that we wish to move to a point which is 15 seconds prior to the last media samples played out. To do this, we will subtract 400 from 240. The resultant value is less than zero. It is an invalid playback point in the current playback unit. Hence, we

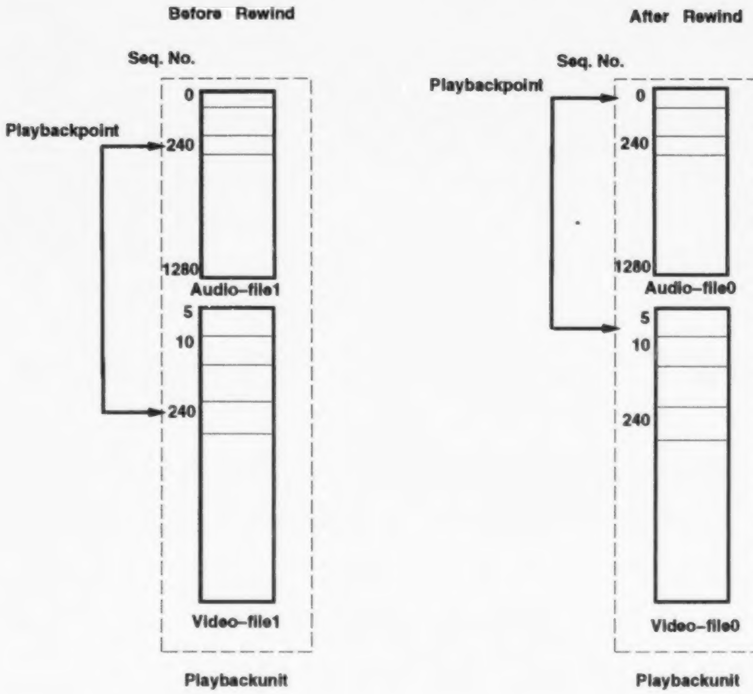

Before Rewind
Seq. No.
Playbackpoint
0
240
1280
Audio-file1
5
10
240
Video-file1
Playbackunit
After Rewind
Seq. No.
Playbackpoint
0
240
1280
Audio-file0
5
10
240
Video-file0
Playbackunit

Figure 76: Rewinding in a multimedia session (case 2)

move to the start of the previous playback unit. In this case, the files *audio-file0* and *video-file0* represent the new playback unit. The playback point in this new playback unit is zero, i.e, we move to the start of the playback unit. Both the audio and video file pointers are adjusted to coincide with this point. This case is illustrated in Fig. 76.

5.5.2 The Fastforward feature

Let the sequence number of the last played out audio data unit in a playback unit be a. This sequence number represents the playback point in the playback unit. If the fastforward option is selected now, we add the value *fastforward-stepsize* to a.

If the resultant value is less or equal to the maximum sequence number value before wraparound, then it represents a valid audio data unit in the playback unit. The audio file pointer is moved to the data unit with this sequence number. This sequence number becomes the new playback point. The video file pointer is also moved to a video data unit with the same sequence number. This is fine scrolling as applied to the fastforward feature. For example, let *audio-file0* and *video-file0* constitute the the present playback unit. Let the sequence number 560 be the playback point. It means that the last audio data unit which was played out had a sequence number of 560. In terms of time units, this corresponds to 21 seconds of playback time from the start of the current playback unit. Let the *fastforward-stepsize* parameter be 400 (corresponding to 15 seconds of playback time) and the maximum sequence number value before wraparound be 1280 (corresponding to 48 seconds of playback time). If a fastforward option is selected at this stage, it means that we want to move to a playback point which is 15 seconds ahead of the last samples which were played out. To do this, we will add the stepsize value of 400 to the current playback point value of 560. The resultant value 960 ($\leq$ 1280) represents a valid sequence number in the playback unit. Both the audio and video file pointers are shifted to coincide with a data unit with a sequence number of 960. The sequence number 960 becomes the new playback point. In effect, we have shifted the playback point 15 seconds ahead of the last media samples played out. This is illustrated in fig. 77.

On the other hand, if the resultant value is greater than the maximum sequence number value, it means that we wish to fastforward to a point which is later than the last playback point in the present playback unit. Here, we resort to coarse scrolling. The playback unit itself is skipped and we move to the next playback unit. For example, let *audio-file1* and *video-file1* constitute the present playback unit. Let the sequence number 1120 (corresponding to 42 seconds of playback time) be the playback point and the maximum sequence number value before wraparound be 1280 (corresponding to 48 seconds of playback time). Let the *fastforward-stepsize* parameter have a value of 400 which corresponds to 15 seconds of audio playback time. If a fastforward option is selected at this stage, it means that we wish to move 15 seconds ahead of the last media samples which were played out. To enable this, we will add the stepsize value of 400 to the playback point value of 1120. The resultant

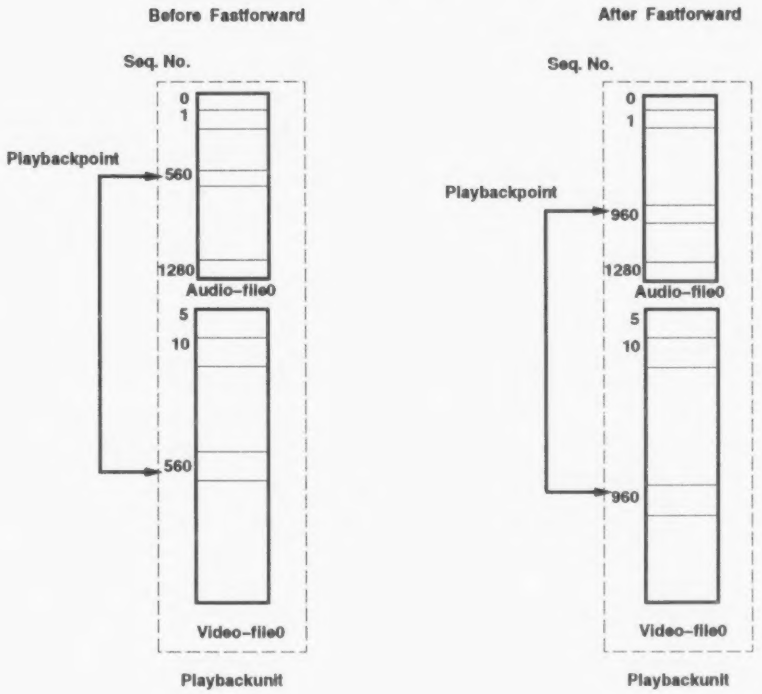

Before Fastforward
Seq. No.
0
1
Playbackpoint
560
1280
Audio-file0
5
10
560
Video-file0
Playbackunit
After Fastforward
Seq. No.
0
1
Playbackpoint
960
1280
Audio-file0
5
10
960
Video-file0
Playbackunit

Figure 77: Fastforwarding in a multimedia session

value is greater than 1280. It is an invalid playback point in the current playback unit. Hence, we move to the start of the next playback unit. In this case, the files *audio-file2* and *video-file2* represent the new playback unit. The playback point in this playback unit is zero, i.e., we are at the start of the new playback unit. Both the audio and video file pointers are adjusted to coincide with this point. This case is illustrated in Fig. 78.

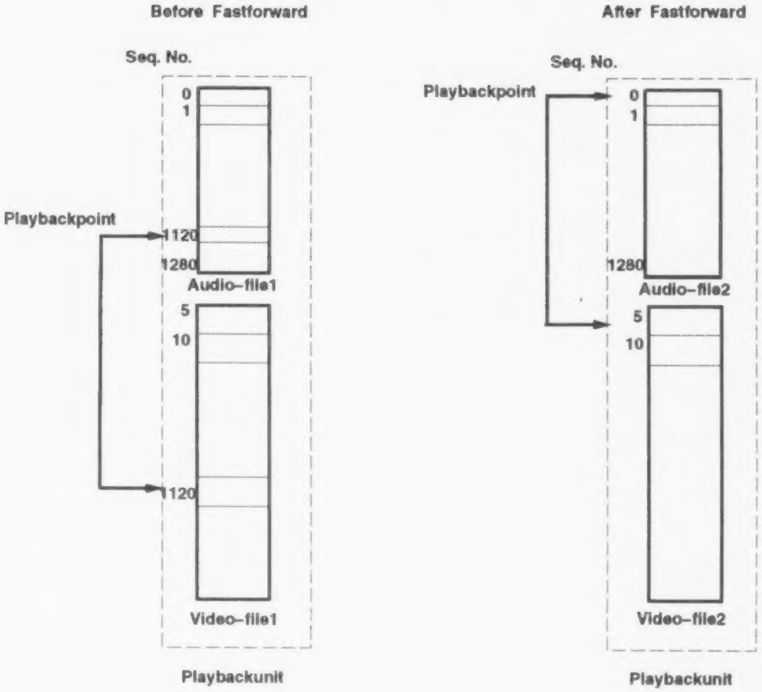

Before Fastforward
After Fastforward
Seq. No.
Seq. No.
Playbackpoint
0
1
1120
1280
Audio-file1
5
10
1120
Video-file1
Playbackunit
Playbackpoint
0
1
1280
Audio-file2
5
10
Video-file2
Playbackunit

Figure 78: Fastforwarding in a multimedia session (case 2)

Chapter 6

Simulating video sessions over ATM

6.1 Performance of SPAFLAY over an ATM network

In this section, we present the simulation results of video sessions over a network of ATM switches. The simulations were done using SMURPH [9]. We are interested in the Quality of Service offered by an ATM network to full motion video sessions. In keeping with the issues discussed in the preceding chapters, parameters like cell loss in the network, end-to-end delay (latency of the application) and jitter are of particular concern to us. SPAFLAY is used as the traffic generator for these video sessions. The network topology, switch architecture and traffic pattern [12] used in these simulations are as follows:

Network topology: The network consists of *end-nodes* and *switches.* The switch is characterized by its connectivity, which represents the number of input and output ports (assumed to be the same), connection set-up and call admission policy, and buffering/policing scheme which describes how cells are buffered and the action taken when the switch runs out of buffer space. Each output port of the switch is assigned a definite transmission rate. This rate is assumed to be an attribute of the (unidirectional) channel (link) connecting the output port to the corresponding input port of another switch. Each ATM channel is represented by a pair of unidirectional links connecting the same pair of switches but in opposite directions.

The end-nodes represent hosts interfaced to the communication subnet. Each end-node is connected via an ATM channel to one switch. The network topology does not change during simulation. The structure of the *virtual paths* (VP's) is assumed to be static over the simulated time interval. Hence, the model does not handle VPI switching. For every end-node reachable from a given switch, the switch maintains a number of routes via which the destination can be reached. The exact route which is to be taken is decided during call-set-up processing, based on the call admission algorithm associated with the switch.

Both data traffic and call-set-up messages originate at the end-nodes and are addressed to end-nodes. The call-set-up messages are processed at intermediate switches. In our simulations, a call-set-up message is never rejected by the switches. This assumption was made so as to enable as many simultaneous video sessions in the network as possible. Call-set-up messages in this SMURPH implementation are single-cell messages.

Another simplification in the implementation is the elimination of explicit VCI switching. For each connection, the VCI (*virtual circuit identifier*) is selected globally from a central pool of available identifiers and used to tag all cells which carry traffic related to the connection. When the connection is set up, every switch along its path sets up an entry in its internal table, which associates ports with the VCI. As this VCI is global, there is no change in the VCI value when a cell is transferred from one switch to another.

Switch architecture: The switch has a pool of buffers associated with every output port. A data cell arriving at the switch is stored at the end of the FIFO list of cells destined for a given output port. If no buffer space is available at the port, the last cell from the list is dropped. The buffering strategy deals with two types of cells called *red* and *green*. Red cells have the CLP bit set to 1, while the green cells have the CLP bit set to 0. Consequently, red cells are considered less critical than green cells and, if there is no room to accommodate a new outgoing cell into a port queue, the switch will try to drop a red cell first. Only if no such cell is available will the switch consider dropping a green cell.

Video traffic generator: Video traffic is generated at the end-nodes. These end-nodes represent hosts, and have only one pair of ports each and do no switching. A typical video teleconference scene was filmed and encoded using the SPAFLAY codec. The *prioritized mode* of operation of SPAFLAY was used with the **Quality** parameter set to 1. The output of the codec (in terms of bits per frame) is used as the VBR video traffic. The peak rate and the average rate of the traffic is calculated and used to drive the simulations.

6.2 Simulation parameters

- The network topology considered for simulation is depicted in Fig. 79. There are 24 end-nodes and 36 4x4 ATM switches arranged in the form of a grid.
- The output port of each switch is assigned a transmission rate of 6.5 Mbps.
- The length of all links is the same: about 10 km.
- Policing of video traffic from the end-nodes is carried out by restricting the traffic rate to either the peak rate or to one half of the sum of the average and the peak rate. In the *prioritized mode* of operation of SPAFLAY (**Quality=1**), the peak rate was found to be 2.9 Mbps, while the average rate was found to be 1.14 Mbps for the session filmed.

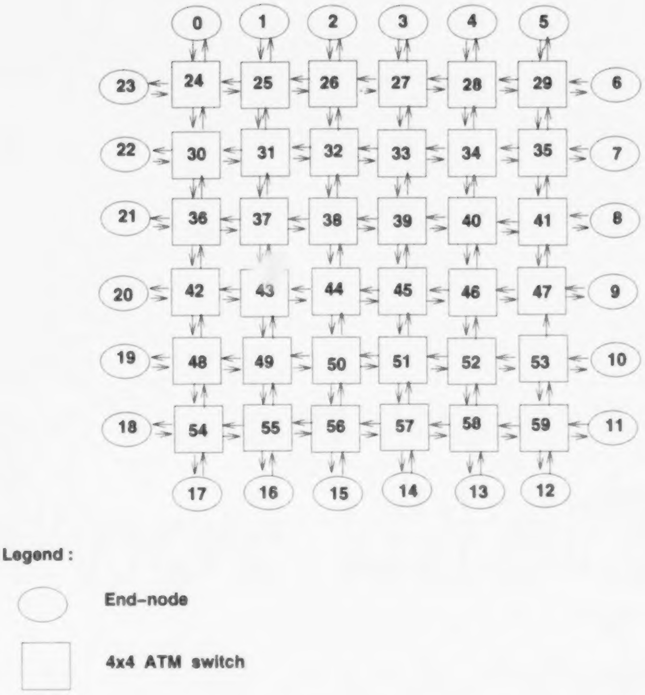

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
Legend :
End-node
4x4 ATM switch

Figure 79: Topology of the network

- 24 video sessions are run simultaneously. End-nodes (0,12), (1,13), (2,14), (3,15), (4,16)......(11,23), (12,0), (13,1), (14,2), (15,3), (16,4)....(23,11) form video source-destination pairs.

6.3 Simulation results

In keeping with the *prioritized mode* of operation in SPAFLAY, the encoded video information is sent in cells which are assigned either a high or low priority value. We vary two parameters in this experiment: the policing policy and the buffer size associated with each ATM switch port. The policing policy either enables peak rate access traffic (**Case1**) or one half of the sum of the peak and the average rate (**Case2**). In each case, we note the cell loss in the network, the end-to-end delay and the effect of Delay Jitter.

From Fig. 80, we find that the cell loss in the network decreases as the buffer size in the switches is increased. This decrease in the cell loss percentage is quite appreciable for the first few buffer size values. We also find that the percentage of high priority cells lost is a fraction of the total cells lost for any given switch buffer value. It reduces to zero when the buffer value is about 300. This figure gives us a measure of the buffer size that we must have at the switches so that we can "guarantee" at least no loss of high priority cells.

If one half of the sum of the average and the peak rate is allocated in the *access network* (**Case 2**), there is less cell loss for a given ATM switch buffer value as compared to **Case 1** where peak rate is allocated in the access network. This is due to the *smoothing* of the traffic that is achieved by restricting the traffic in the subnet to less than the peak rate. This smoothing is typically done by buffering cells at the UNI (User-Network Interface). It helps to lower the percentage of cell loss in the ATM network at the expense of increasing the end-to-end delay. This is seen from Fig. 81 which shows the average end-to-end frame delay values for the 24 active sessions.

The effect of Delay Jitter is studied as follows: We assume every end-node which effects playback of these video frames to have a finite buffer called "playback buffer". Cells which arrive at the receiver's end-node via the network are stored in the playback buffer.

Playback of the first frame begins when the playback buffer is half full. Subsequent frames are played out every 1/15 seconds which is the same as the generation time interval of video frames. At a playback point, we play out a complete video frame present in the playback buffer or portions of the frame if the entire frame has not arrived. Every frame is identified uniquely on the basis of the sequence number which it carries. At each playback point, the receiver plays out the next expected video frame in the sequence.

Jitter can affect this playback schedule in two ways: Cells (corresponding to a particular frame) may miss their playback point because they are "late". These cells are as good as lost and cannot be played out. This constitutes one form of "playback loss".

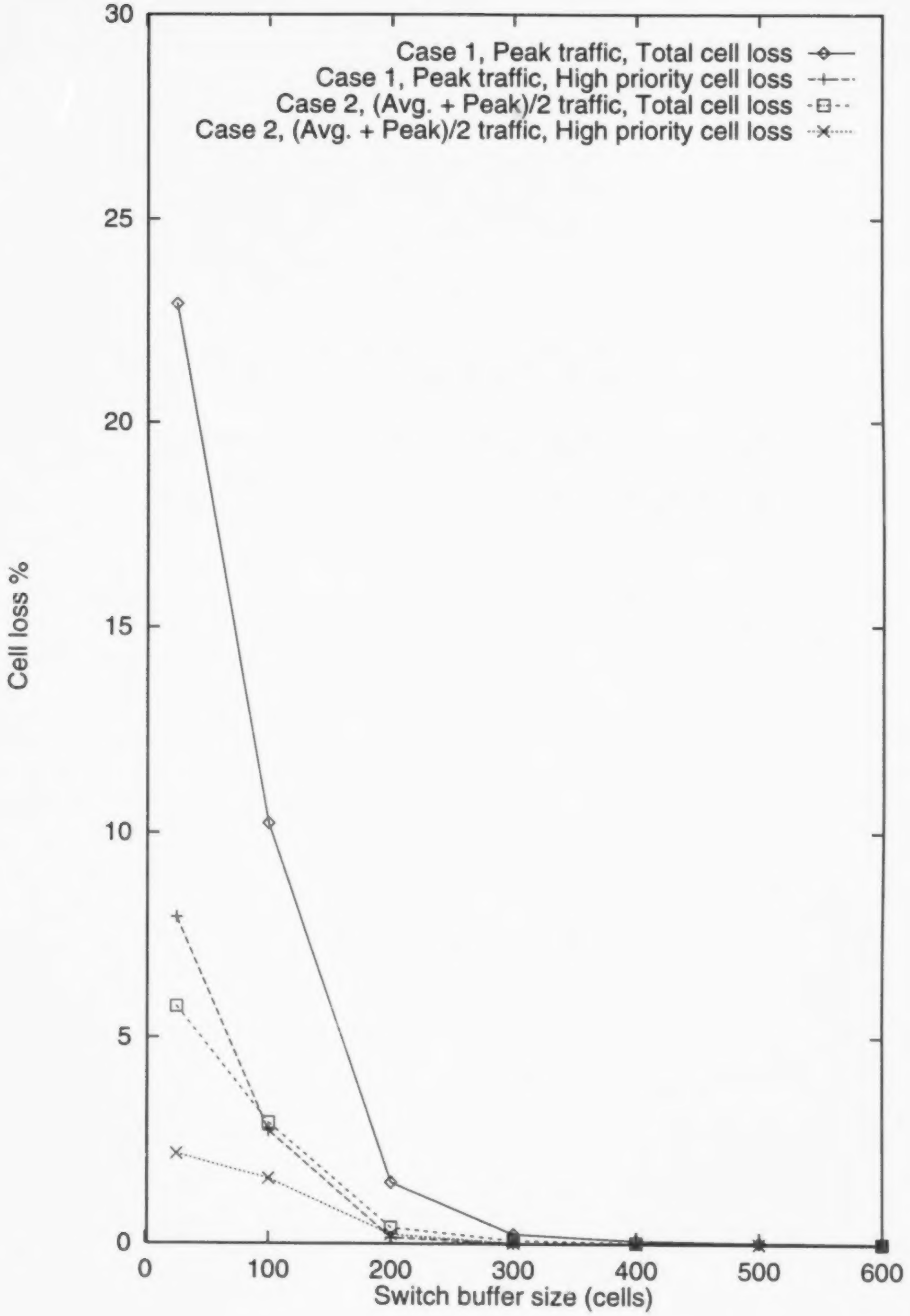


Figure 80: Cell loss in the ATM network

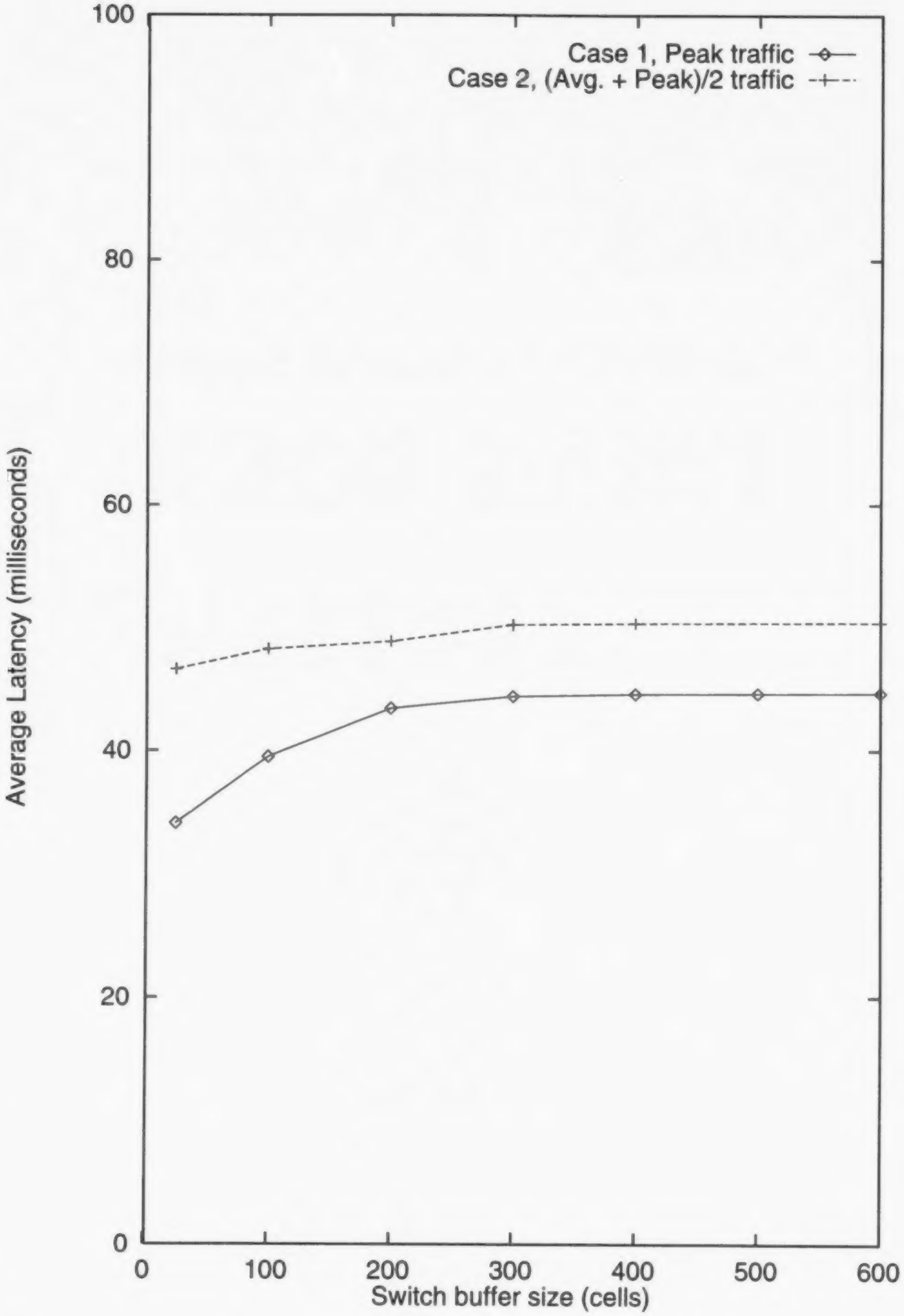


Figure 81: Average end-to-end frame delay (Latency)

Network jitter may also cause frames to get "clumped" in the network due to the variable delays incurred in the switch buffers. In this case, the cells corresponding to these frames arrive at the receiver separated by smaller time intervals than their generation time intervals. This causes the playback buffer to overflow leading again to "playback loss".

As the playback buffer size is increased, we minimize the playback loss due to "late" frames and "clumped" frames. Hence, our study of Delay Jitter essentially reduces to the problem of identifying a playback buffer size of adequate capacity.

For each policing strategy and network switch buffer size, we vary the size of the playback buffer. We measure the percentage of playback loss in each case. Fig. 82 shows the playback cell loss percentage for increasing values of playback buffer size. The access traffic is kept at the peak value in these experiments. As anticipated, the playback cell loss percentage decreases as the playback buffer size is increased. The cell loss percentage is almost 0% when the playback buffer size is about 600 cells. Another observation is that the playback cell loss for a given value of the playback buffer is more when the size of the switch buffers in the network is increased. As we increase the switch buffer size in the network, we increase the possibility of queuing delays in the network. This causes more jitter and hence an increase in the playback cell loss.

Fig. 83 shows the playback cell loss percentage for increasing values of playback buffer size. Here, the access traffic is kept at one half of the sum of the average and the peak value. The playback cell loss percentage again decreases as the playback buffer size is increased. The cell loss percentage is almost 0% when the playback buffer size is about 600 cells. We again find the trend of increased playback cell loss when the size of the switch buffers in the network is increased.

Fig. 84 compares the effect on playback cell loss when the two access traffic policies are used. The switch buffer size is kept constant at 25 cells. We vary the playback buffer size and the access traffic rate. We notice that dedicating peak traffic in the access network decreases the jitter and hence the playback cell loss. This effect is also observed for different values of the switch buffer size as shown in Figs. 85 to 89.

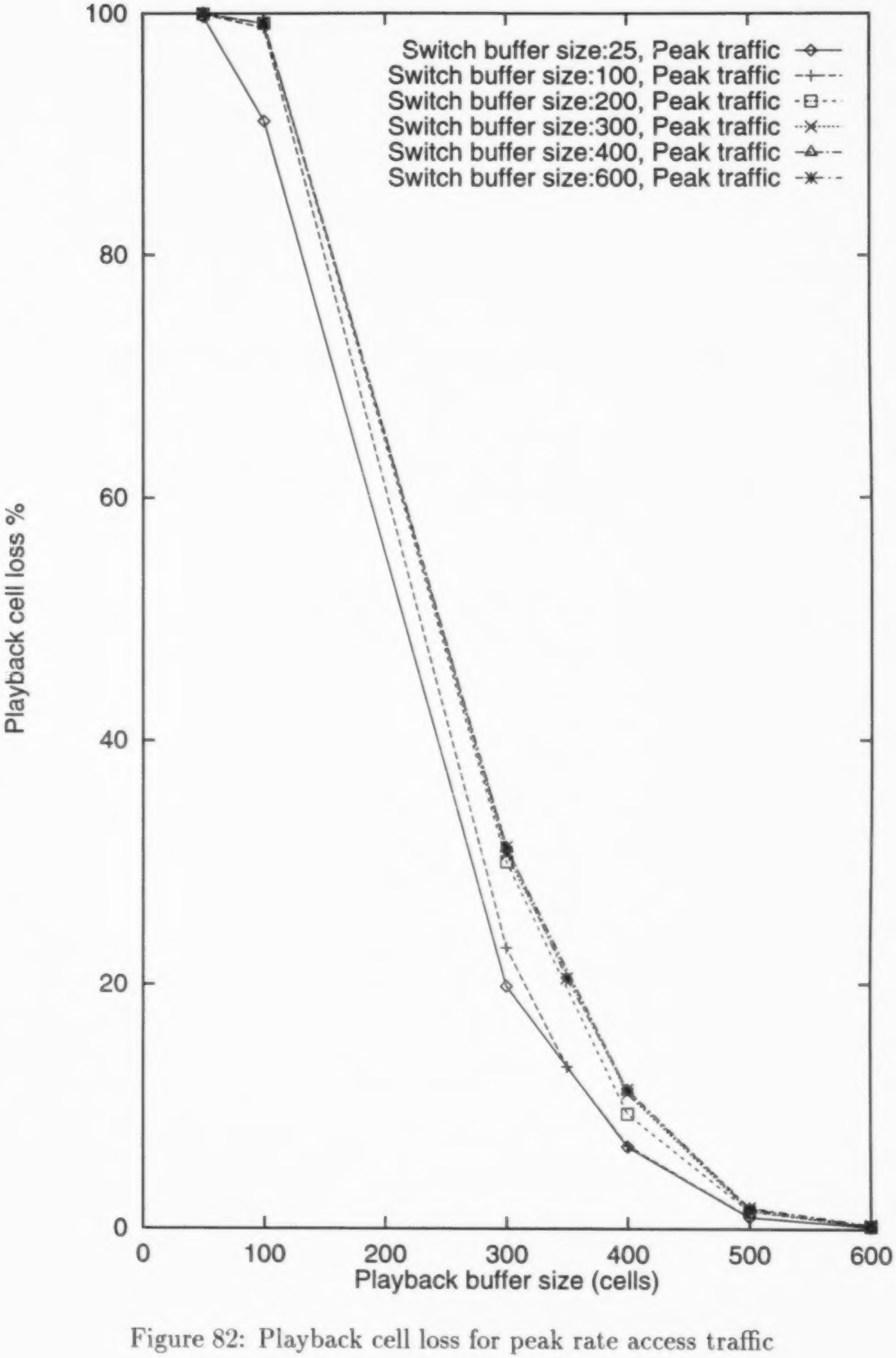


Figure 82: Playback cell loss for peak rate access traffic

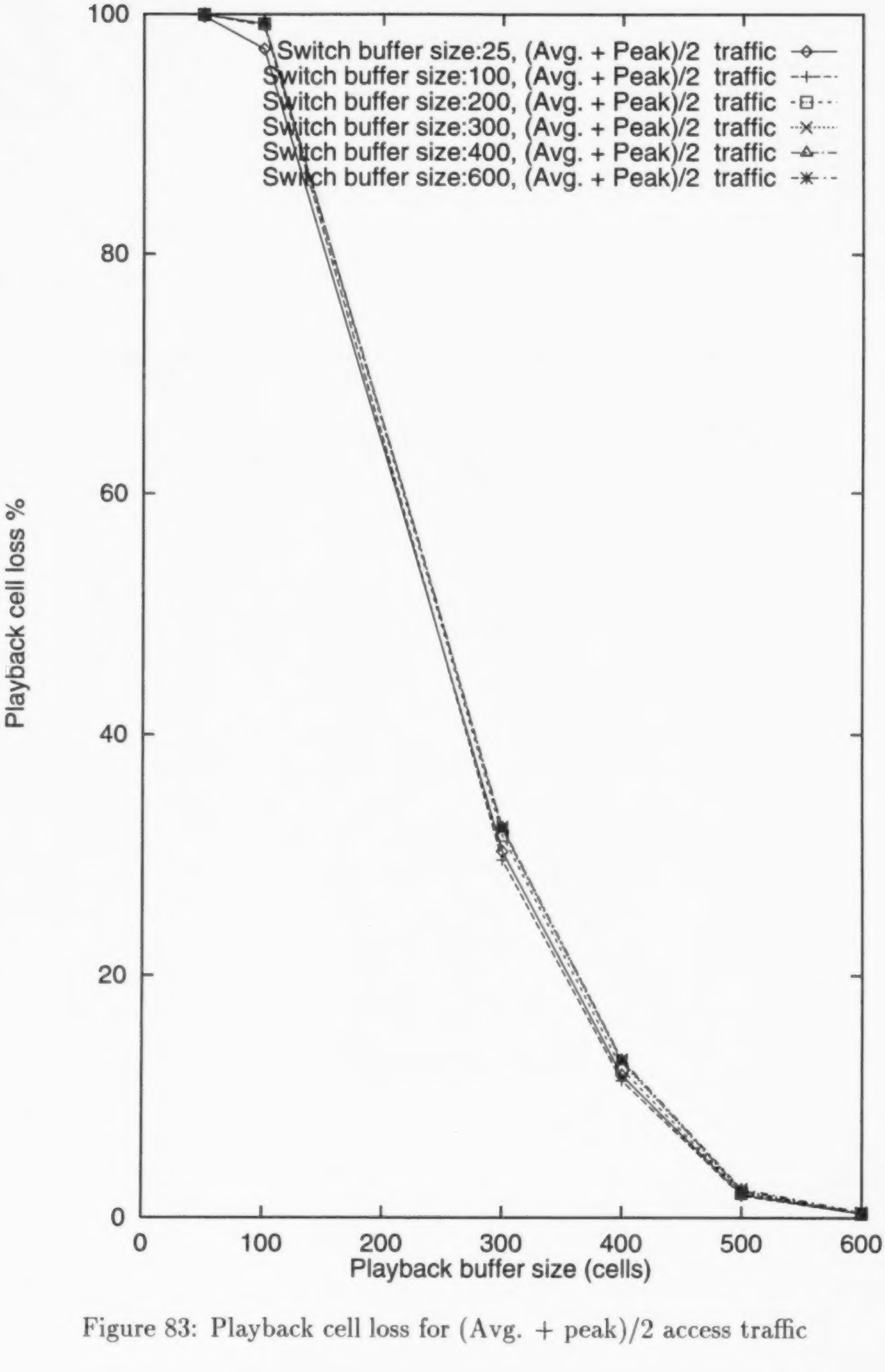


Figure 83: Playback cell loss for (Avg. + peak)/2 access traffic

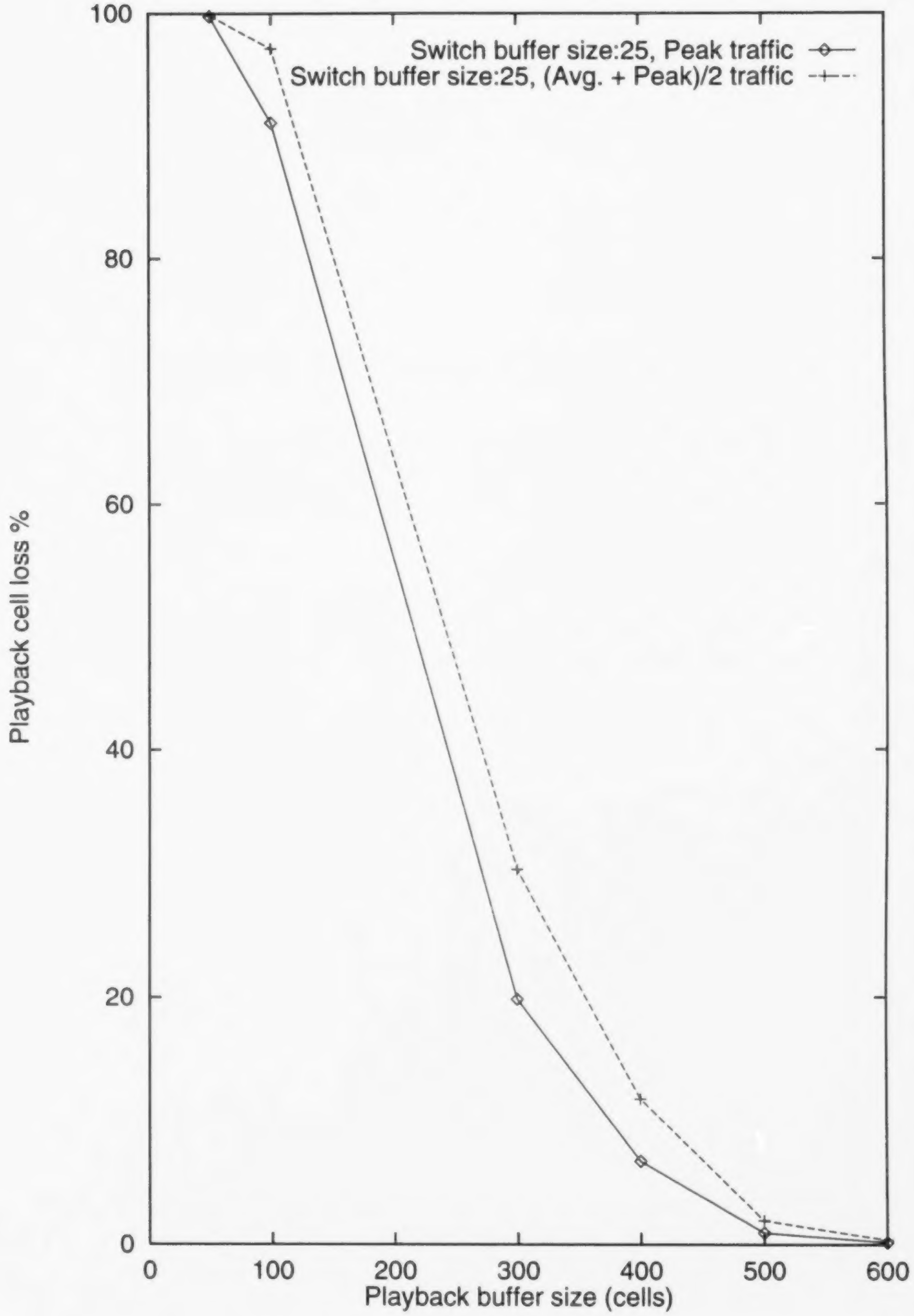


Figure 84: Switch buffer size=25: Playback cell loss

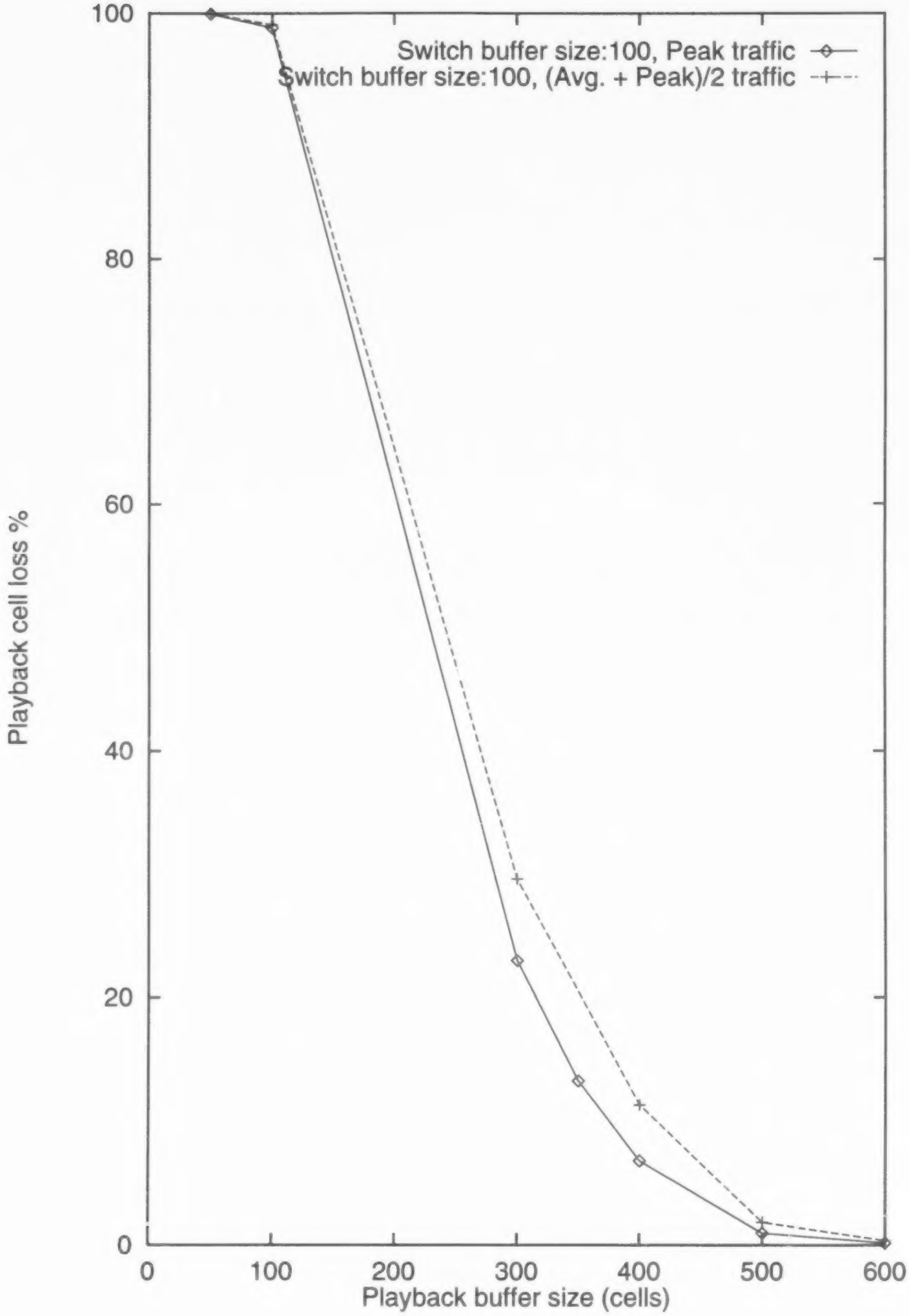


Figure 85: Switch buffer size=100: Playback cell loss

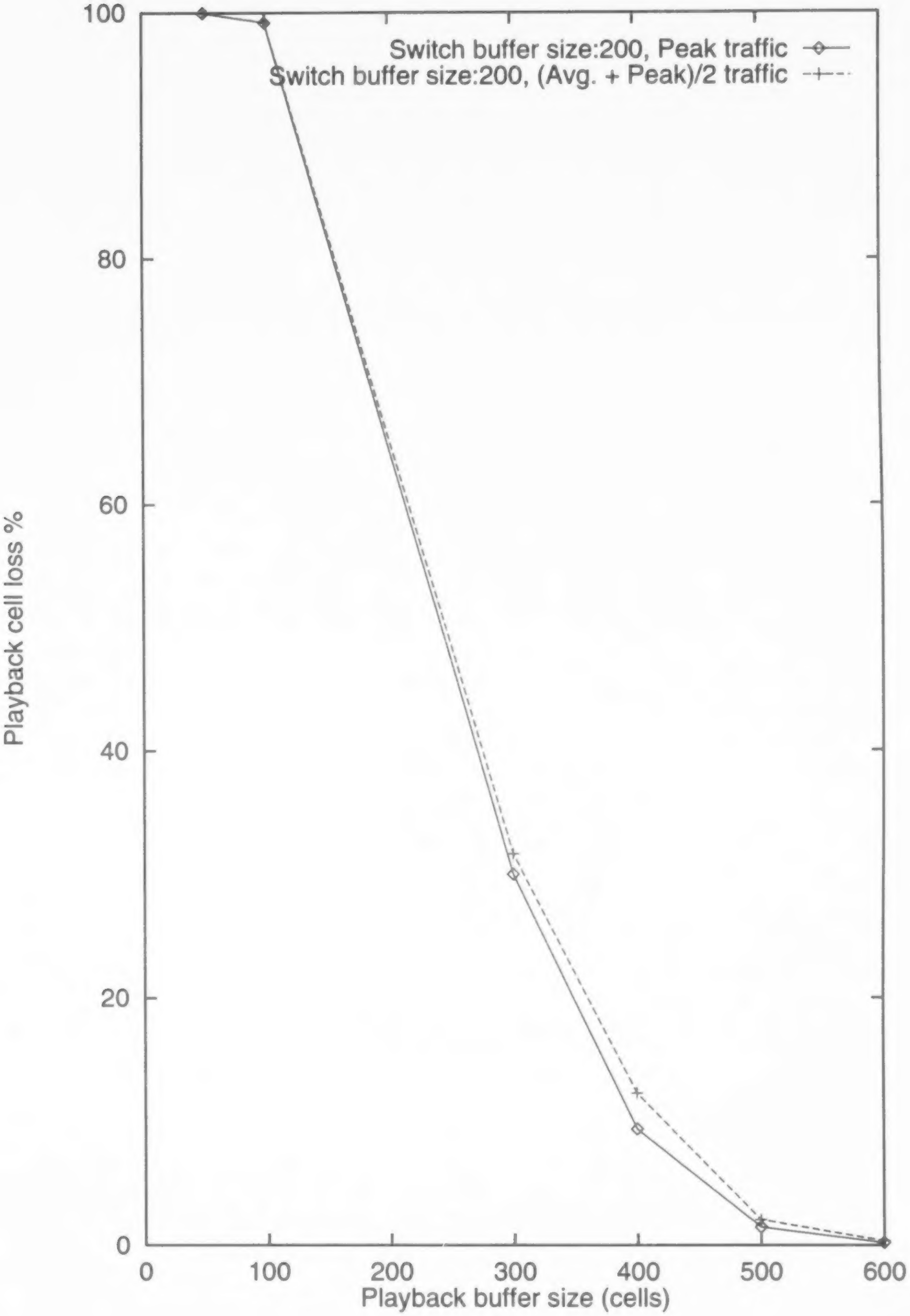


Figure 86: Switch buffer size=200: Playback cell loss

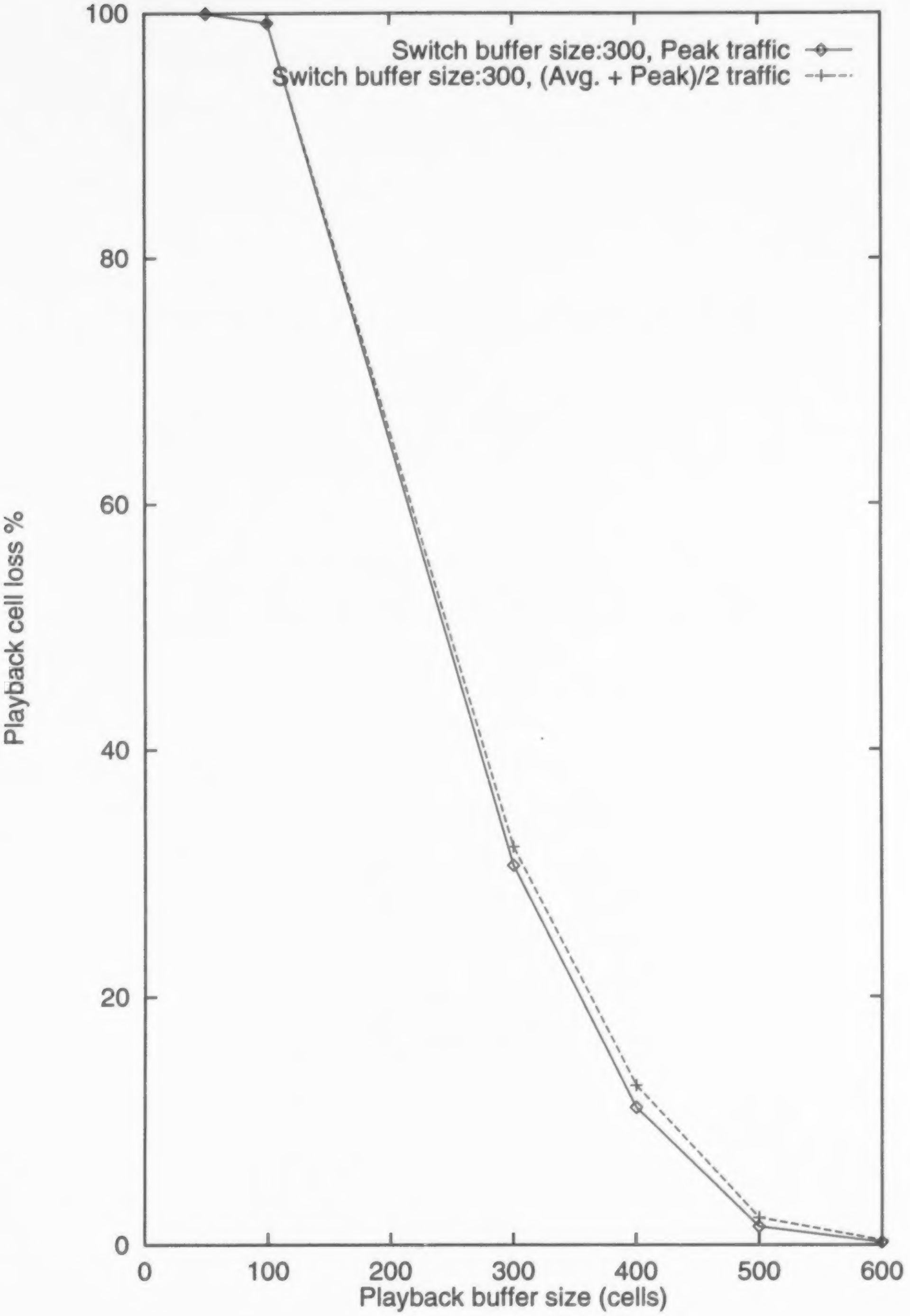


Figure 87: Switch buffer size=300: Playback cell loss

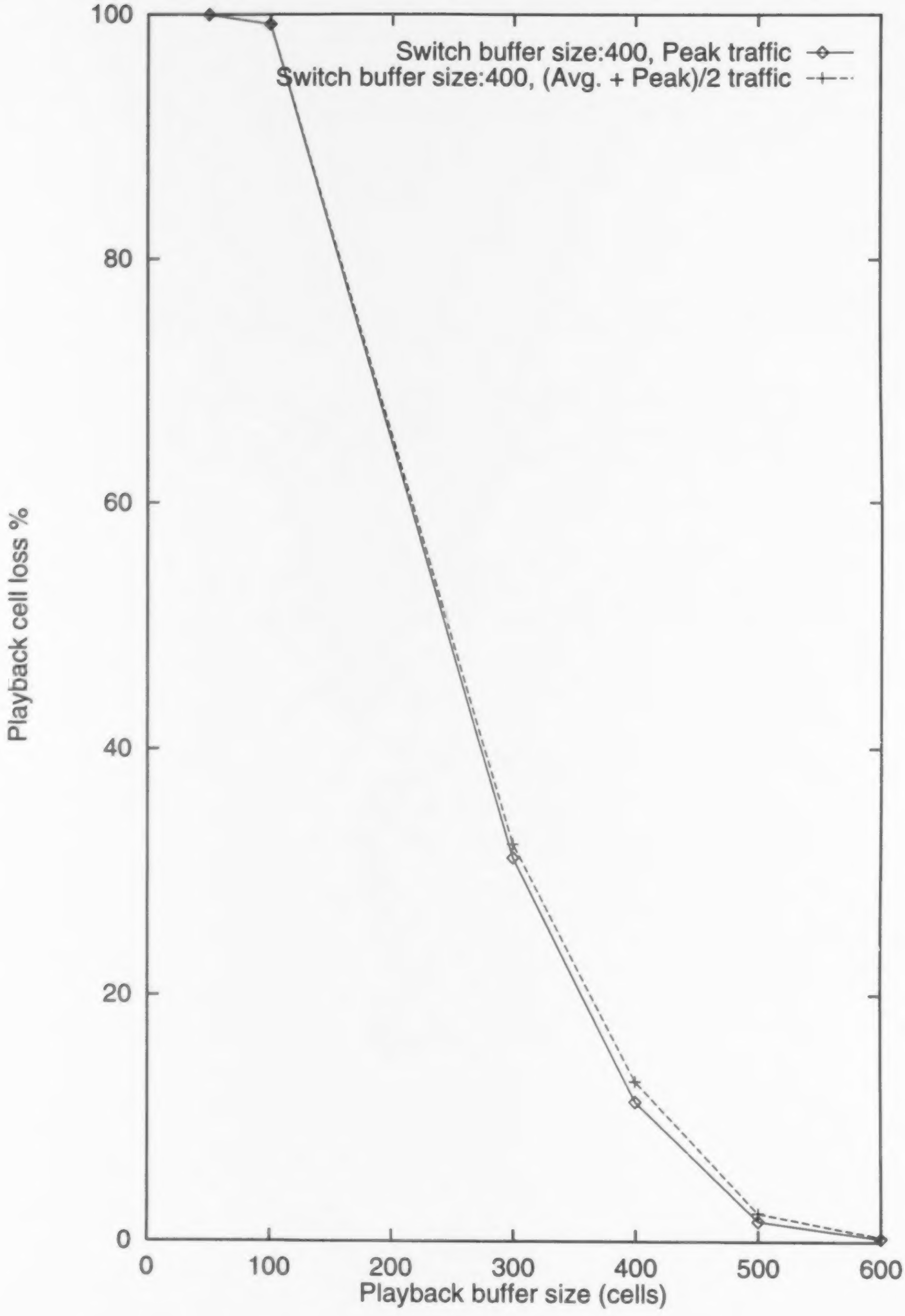


Figure 88: Switch buffer size=400: Playback cell loss

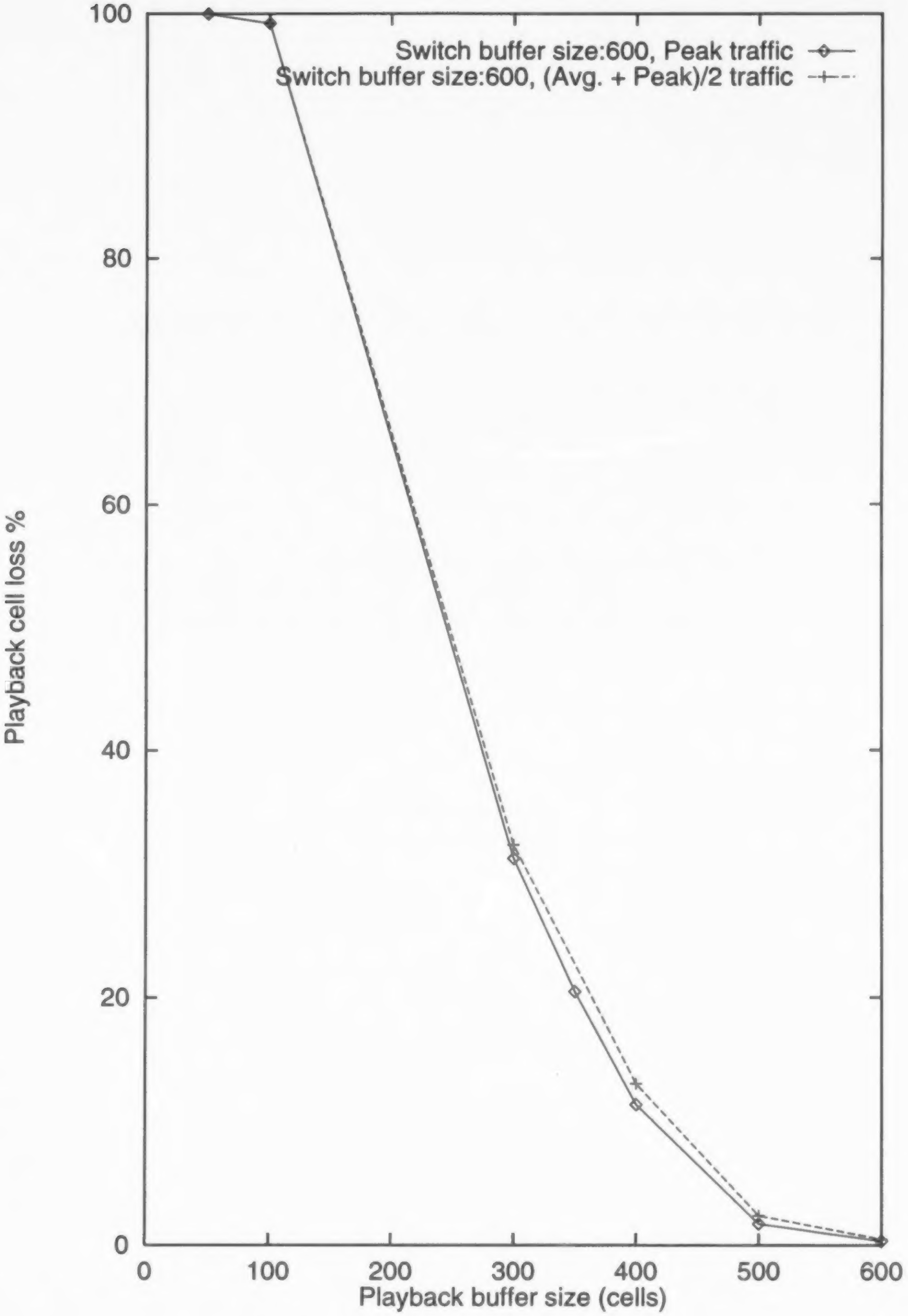


Figure 89: Switch buffer size=600: Playback cell loss

Chapter 7

Conclusion and future directions

We have discussed the design of a video codec suited to ATM networks in this thesis. SPAFLAY uses a combination of spatial and frequency domain compression techniques. We also use layering in the spatial and frequency domain to categorize information into high priority and low priority groups. Using this scheme helps to reduce the subjective deterioration of the resultant image quality in the event of cell loss in the ATM network. The inbuilt prioritization scheme in SPAFLAY is particularly suited to teleconferencing applications and in a multicast scenario.

The layered coding algorithm used in SPAFLAY increases the robustness of the video codec to cell loss. However, the advantage is at the expense of incurring a high bit-rate-penalty compared to one-layer approaches. We have provided a quantitative measure of the extent of bit-rate-penalty by comparing SPAFLAY's *prioritized mode* of operation against the *normal mode* of operation.

Synchronization of multiple data streams in time has been recognized as a significant requirement of future multimedia applications utilizing broadband communication technology. We have considered synchronization of audio and video streams at the application level in this thesis. Two intra-medium synchronization techniques: Blind-Timing and Absolute-Timing have been used. A master-slave-based scheme is used for inter-media synchronization.

We have discussed the functionality of the application which uses the SPAFLAY video codec and the media synchronization techniques. This application can be used to playback live as well as stored audio and video data. The application can be used to playback sessions in a unicast as well as in a multicast mode. Simulation results are used to arrive at a reasonable playback buffer size to compensate for jitter over the network.

7.1 Future directions

- Extending the SPAFLAY video codec to support multiple, moving foveae instead of a single fovea.
- Using multiple priority levels instead of a two-level priority scheme to compartmentalize video information.
- Extending SPAFLAY to work in an adaptive mode, where the area of spatial importance in a video frame is dynamically varied depending on network conditions.
- Using a combination of DCT and Haar-based Combined Transform Coding techniques instead of using Haar alone. DCT effects more compression but is computationally more expense than Haar. It might be advantageous to switch between these two transform coding techniques in a dynamic fashion.
- Associating more slave media (images, text, etc.) with the master medium audio stream.

Bibliography

[1] Anderson D. P., and Homsy G., "A Continuous Media I/O Server and Its Synchronization Mechanism," *Computer,* Vol. 24, No. 10 October 1991, pp 51-57.

[2] Andrews H. C., *Computer Techniques in Image Processing,* Academic Press, 1970.

[3] Basu A., and Wiebe K., "Videoconferencing using Spatially Varying Sensing with Multiple and Moving Foveae," IEEE Proceedings of International Conference on Pattern Recognition, Jerusalem, Israel, October 1994, (TRlabs/NSERC IOR Project).

[4] Basu A., Jain M., and Li X., "Variable-Resolution Boundary Detection," Pattern Recognition Letters, in press, 1995.

[5] Bertsekas D., and Gallager R., *Data Networks,* Prentice-Hall.

[6] Clarke R. J., *Transform Coding of Images,* Academic Press, 1985.

[7] Cohen D. M. et.al., "Performance Modeling of Video Teleconferencing in ATM Networks," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 3, No. 6, December 1993, pp 408-420.

[8] Didier Le Gall., "MPEG: A Video Compression Standard for Multimedia Applications," Communications of the ACM, Vol. 34, No. 4, April 1991, pp 47-58.

[9] Gburzynski P., "Protocol Design for Local and Metropolitan Area Networks," Prentice Hall, 1995(to appear).

[10] Deering S., "IP Multicast Extensions for 4.3 BSD UNIX and related systems," MULTICAST 1.2 Release.

[11] Deering S., "Multicast Routing in a datagram internetwork," A dissertation submitted to Stanford University.

[12] Dobosiewicz W., and Gburzynski P., "Modeling ATM networks: a case study," MASCOTS'95, SCS + IEEE + ACM + IFIP, Durham, North Carolina, January 18-20, 1995.

[13] Fox E. A., "Advances in Interactive Digital Multimedia Systems," *Computer,* Vol. 24, No. 10, October 1991, pp 9-19.

[14] Ghanbari M., "An adapted H.261 Two-Layer Video Codec for ATM Networks," IEEE Transactions on Communications, Vol. 40, No. 9, September 1992, pp 1481-1490.

[15] Hall E. L., *Computer Image Processing and Recognition,* Academic Press, 1979.

[16] Hosoda K., "Variable Rate Video Coding Scheme for Broadcast Quality Transmission and its ATM Network Applications," IEICE Trans. Commun., Vol. E75-B, No. 5, May 1992, pp 349-357.

[17] Huitema C., et.al, "Software codecs and works station video conferences," INRIA, Sophia-Antipolis, France, November 1993.

[18] Joseph K., et.al, "Design and Performance Evaluation of a Variable Bit-Rate (VBR) Video Compression Algorithm for ATM Networks," GLOBECOM '91, pp 9-15.

[19] Kinoshita T., et.al., "Variable-Bit-Rate HDTV CODEC with ATM-Cell-Loss Compensation," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 3, No. 3, June 1993, pp 230-237.

[20] Kishimoto R., et.al., "Generation Interval Distribution Characteristics of Packetized Variable Rate Video Coding Data Streams in an ATM Network," IEEE Journal on Selected Areas in Communications, Vol. 7, No. 5, June 1989, pp 833-841.

[21] Kishino F., et.al., "Variable Bit-Rate Coding of Video Signals for ATM Networks," IEEE Journal on Selected Areas in Communications, Vol. 7, No. 5, June 1989, pp 8801-806.

[22] Little T. D., and Ghafoor A., "Spatio-Temporal Composition of Distributed Multimedia Objects for Value-Added Networks," *Computer,* Vol. 24, No. 10 October 1991, pp 42-50.

[23] Little T. D., and Ghafoor A., "Multimedia Synchronization Protocols for Broadband Integrated Services," IEEE Journal on Selected Areas in Communications,Vol. 9, No. 9, December 1991, pp 1368-1381.

[24] Minoli D., and Keinath R., "Distributed Multimedia Through Broadband Communications," Artech House, 1993.

[25] Murakami H., et.al., "Considerations on ATM Network Performance Planning," IEICE Trans. Commun., Vol. E75-B, No. 7, July 1992.

[26] Nicolaou C., "An architecture for Real-Time Multimedia Communication Systems," IEEE Journal on Selected Areas in Communications, Vol. 8, No. 3, April 1990, pp 391-400.

[27] Pancha P., et.al., "A look at the MPEG video coding standard for variable bit rate video transmission," INFOCOM 1992, pp 85-94.

[28] Pancha P., et.al., "Bandwidth-Allocation Schemes for Variable-Bit-Rate MPEG Sources in ATM networks," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 3, No. 3, June 1993, pp 190-198.

[29] Pennebaker W. B., and Mitchell J. L., *JPEG Still Image Data Compression Standard.* Van Nostrand Reinhold.

[30] Ramanathan S., et.al, "Integrating Virtual Reality, Teleconferencing, and Entertainment into Multimedia Home Computers," IEEE Transactions on Consumer Electronics, Vol. 38, No. 2, May 1992, pp 70-75.

[31] Ravindran K., "Transport Models for Synchronization of Multimedia Data streams," Technical Report, TR:93-8, January 1993.

[32] Robert J. W., "Variable-Bit-Rate Traffic Control in B-ISDN," IEEE Communications Magazine, September 1991, pp 50-56.

[33] Russel J., "Multimedia Networking Performance Requirements," ATM Networks, I. Viniotis and R. O. Onvural (eds.), New York:Plenum Pubs., pp. 187-198.

[34] Schulzrinne H., and Casner S., "RTP: A Transport Protocol for Real-Time Applications," Internet Engineering Task Force INTERNET-DRAFT, 1993.

[35] Schulzrinne H., "Issues in Designing a Transport Protocol for Audio and Video Conferences and other Multiparticipant Real-Time Applications," Internet Engineering Task Force INTERNET-DRAFT, 1993.

[36] Shenker Scott., "A service Model for an Integrated Services Internet" Internet Engineering Task Force INTERNET-DRAFT, 1993.

[37] Steinmetz Ralf., "Synchronization Properties in Multimedia Systems," IEEE Journal on Selected Areas in Communications, Vol. 8, No. 3, April 1990, pp 401-405.

[38] Tripathi S., "Issues in Distributed Multimedia Applications," University of Maryland.

[39] Verbiest Willem., "A Variable Bit Rate Video Codec for Asynchronous Transfer Mode Networks," IEEE Journal on Selected Areas in Communications, Vol. 7, No. 5, June 1989, pp 761-770.

[40] Zhang Y. Q., et.al, "Variable-Bit-Rate Video Transmission in Broadband-ISDN Environment," IEEE Trans. Commun., pp 213-222.